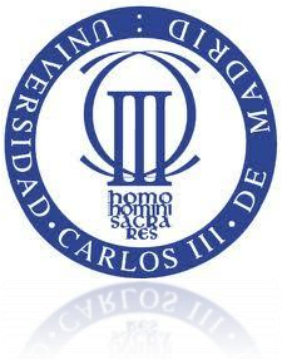


Universidad Carlos III de Madrid
Escuela Politécnica Superior
Ingeniería en Informática

Proyecto Fin de Carrera

IOPFS: Sistema de ficheros basado servidores intermedios de almacenamiento

Autor: Alberto Valle Amigo
Tutor: Luis Miguel Sánchez García
Septiembre 2011

Agradecimientos

A mis padres y mi hermana, por haberme apoyado desde el inicio de esta carrera y sin los cuales no podría haber llegado hasta aquí.

A Cristina, mi novia por todo su apoyo durante la carrera y en especial durante el desarrollo del Proyecto final de Carrera.

A Diego, mi compañero de carrera, por las prácticas interminables y las prisas de última hora.

A todos mis amigos, por interesarse y apoyarme.

A mi tutor de proyecto Luis Miguel, por haber estado disponible en todo momento y por su gran ayuda.

Índice de contenidos

Índice de figuras

Índice de diagramas

Índice de tablas

Índice de Gráficos

1. Definición y acrónimos

A continuación, se definen las palabras técnicas y acrónimos que se utilizarán en el documento, facilitando la comprensión del mismo.

1.1 Definiciones

- **Tamaño de bloque**: Tamaño de la unidad de reparto, es decir, la cantidad de datos de un fichero que se almacena en cada subfichero.
- **Política de distribución**: Algoritmo de distribución de los datos de un fichero sobre los servidores.
- ***Benchmark***: técnica utilizada para medir el rendimiento de un sistema o componente de un sistema.
- ***Thread***: proceso ligero que permite a una aplicación realizar varias tareas simultáneamente, los distintos procesos ligeros comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situaciones de autentificación, etc.
- ***Kernel* de Linux**: parte fundamental del sistema operativo, que provee los servicios básicos.
- ***Round-Robin***: es un método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.
- ***Clusters***: conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.

- ***Background***: tarea que es ejecutada en un proceso ligero, no es necesario que la tarea principal espere a que termine la tarea en background.
- ***Backup***: copia de seguridad, con el fin de que se pueda recuperar la información en caso de que falle el sistema principal.
- **Bloque limpio**: bloque de datos almacenado en la base de datos sobre el que no se ha ejecutado ninguna operación de escritura, el bloque contiene la misma información que el bloque en disco.
- **Bloque sucio**: bloque de datos almacenado en la base de datos sobre el que se ha ejecutado alguna operación de escritura, el bloque contiene distinta información que el bloque en disco.
- **Dirección Ip**: es una etiqueta numérica que identifica, de manera lógica y jerárquica, a una interfaz de un ordenador dentro de una red que utiliza el protocolo IP.

1.2 Acrónimos

- **DB**: *Data Base*
- **DNS**: *Domain Name System*
- **E/S**: entrada/salida (I/O)
- **FIFO**: *First In First Out*
- **FUSE**: *Filesystem in Userspace*
- **GFS**: *Google File System*
- **GPFS**: *General Parallel File System*
- **I/O**: *input/output* (E/S)
- **IOP**: *Input/Output Proxy*
- **MDS**: *Meta data server*
- **MPP**: *Massively Parrallell Processor*
- **NFS**: *Network File System*
- **OSS**: *Object storage server*
- **OST**: *Object storage target*
- **PVFS**: *Parallel Virtual File System*
- **P2P**: *Peer to Peer*
- **RAID**: *Redundant Array of Independent Disk*
- **RAM**: *Random Access Memory*
- **RPC**: *Remote Procedure Call*
- **SFD**: Sistema de Ficheros Distribuidos.
- **SFP**: Sistema de Ficheros Paralelo

2. Introducción

En este apartado, se detallarán el contexto, la motivación y los objetivos del Proyecto Fin de Carrera, así como una breve descripción del mismo.

2.1 Contexto

En la actualidad, tanto usuarios como aplicaciones cada vez demandan el uso de un **mayor volumen información**. Normalmente esta información debe estar **almacenada de forma fiable** y para cumplir este fin se hace uso de un sistema de ficheros.

La problemática de tratar con un gran volumen de información ya se ha presentado antes, en entornos de computación de altas prestaciones como simulaciones científicas o las dedicadas a la minería de datos. Estos entornos requieren manejar grandes recurso de cómputo y memoria.

La arquitectura *cluster* se han convertido en la solución más común para solventar esta demanda. Un *cluster* es un conjunto de ordenadores independientes pero interconectados entre sí que trabajan conjuntamente como un único recurso para resolver un problema común.

Figura 1. Cluster – Barcelona Supercomputer Center

Esta arquitectura se ha visto beneficiada por los avances en la tecnología y arquitectura de los microprocesadores, que han mejorado sus prestaciones. La mejora en el almacenamiento de datos también ha ayudado en la mejora de los clusters. Y en especial la mejora de los protocolos y redes de comunicación.

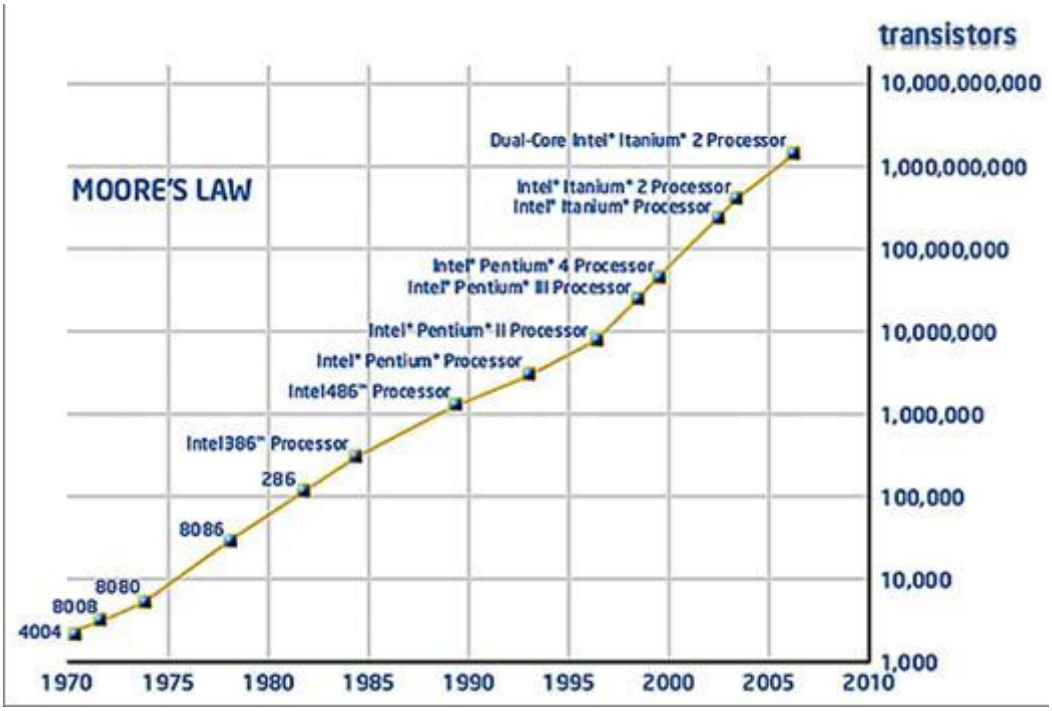


Figura 2. Ley de Moore

Estas mejoras han permitido evolucionar del uso de supercomputadores, que tenían un alto coste, al uso de *clusters*.

A pesar de la evolución de los microprocesadores y de los dispositivos de almacenamiento, estos últimos han evolucionado más lentamente que los primeros. Los dispositivos de almacenamiento tienen tiempos de acceso elevados, que penalizan este tipo de arquitecturas.

Algunos dispositivos de almacenamiento como los discos SSD han reducido los tiempos de acceso.

Otros problemas que presenta esta arquitectura son:

- Costosa administración y gestión del sistema.
- Cuello de botella en la red de comunicación.
- Falta de escalabilidad de los sistemas de almacenamiento.

2.2 Motivación

Google ha desarrollado un sistema de ficheros GFS (*Google File System*), que es capaz de **trabajar con un gran volumen de clientes** accediendo simultáneamente a un **gran volumen de información**. Esto lo logra teniendo la información distribuida en diferentes nodos. Con esto logra que las operaciones de lectura y escritura no converjan en un solo servidor. Este sistema posee **toleración a fallos**, como la caída de un nodo. Los nodos replican su información en otros nodos, con esto se evita la perdida de datos.

Se pretende descentralizar los datos de un único servidor, lo que permitirá reducir los tiempos de comunicación entre nodos. La forma de conseguirlo es introduciendo nuevos nodos en la red y derivando las llamadas E/S a ellos.

Reducir el tiempo de acceso a los datos. Almacenando los datos de los nuevos nodos en memoria, se consigue incrementar la velocidad de las peticiones de lectura y escritura.

Se desea un sistema flexible y fácil de implantar. Se quiere minimizar al máximo el impacto del nuevo sistema de ficheros en la arquitectura y software actual. Para ello el sistema de ficheros se creara como una capa nueva al sistema de ficheros actual.

2.3 Objetivos

Se desea desarrollar un proyecto capaz de **reducir los cuellos de botella** que producen los sistemas de ficheros en red, como NFS. Los objetivos principales del proyecto son:

- Estudiar otros **sistemas de ficheros distribuidos/paralelos** como GFS.
- Estudiar el protocolo de comunicación **RPC** (*Remote Procedure Call*).
- Estudiar la base de datos **BerkeleyDB** con almacenamiento de datos en memoria y en disco.
- Estudiar el sistema de ficheros **FUSE** (*Filesystem in Userspace*), y aplicaciones realizadas usando este sistema.
- **Desarrollar un sistema de ficheros** que reduzca los cuellos de botella sobre NFS.
- Evaluar el funcionamiento de la aplicación.

El sistema de ficheros contará con 3 tipos de nodos:

- **Nodo cliente**: realizará las operaciones E/S sobre los IOP.
- **Nodo de metadatos**: contendrá la información de los IOP.
- **Nodo IOP**: almacenará los bloques de datos.

Los IOP se agruparán formando una red a la que se conectarán los clientes para realizar las operaciones E/S. El sistema será tolerante a fallos y permitirá realizar lecturas y escrituras paralelas permitiendo mejorar los tiempos de lectura y escritura. El sistema permitirá 3 políticas de lectura y 3 políticas de escritura, para adaptarse a las necesidades de cada sistema. Los bloques de datos almacenados en cada IOP podrán almacenarse en memoria o en disco, esto facilitara la instalación de los IOP en máquinas con gran almacenamiento en disco o máquinas con gran cantidad de memoria RAM (*Random Access Memory*). El sistema tendrá una única política de distribución de bloques.

2.4 Estructura del documento

En este apartado se detalla cada uno de los capítulos de los que se compone este documento.

En el capítulo 1, ***INTRODUCCIÓN***, se encuadra el presente proyecto, proporcionando un breve resumen y los objetivos del mismo.

En el capítulo 2, ***ESTADO DE LA CUESTION***, se proporciona una visión general de la tecnología que engloba el proyecto.

En el capítulo 3, ***ANALISIS Y DISEÑO***, se realizara un análisis detallado de los requisitos del sistema y el diseño del mismo.

En el capítulo 4, ***RESULTADOS***, se mostrara una evaluación descriptiva y grafica sobre los resultados obtenidos.

En el capítulo 8, ***CONCLUSIONES Y TRABAJOS FUTUROS***, se realizará un balance de los objetivos logrados y se expondrán las conclusiones obtenidas de la realización del proyecto.

Finalmente, se incluirá un apéndice I, denominado "**MANUAL DE INSTALACIÓN"**, en el que se especificará como realizar la instalación del proyecto, así como los pasos a seguir para realizar la instalación de forma adecuada, otro apéndice II, denominado "**CONFIGURACIÓN**", en el que se explicará como configurar los diferentes sistemas del proyecto, poniendo un ejemplo de una configuración y por último un apéndice III, denominado "**MANUAL DE USUARIO**", el que se detalla cómo realizar la compilación y ejecución del proyecto.

3. Estado de la cuestión

A lo largo de esta sección se detallara el marco del proyecto y tecnologías utilizadas para el desarrollo del mismo.

3.1 Sistema de ficheros

Un sistema de ficheros es un conjunto de datos abstractos, concretamente algoritmos y estructuras lógicas, utilizados para poder acceder a la información que tenemos en el disco y que son implementados para el almacenamiento, la organización jerárquica, la manipulación, el acceso, el direccionamiento y la recuperación de datos. En este apartado se trataran diferentes tipos de sistema de ficheros distribuidos y paralelos

3.1.1 Sistema de ficheros distribuidos

Los sistemas de ficheros distribuidos ofrecen un espacio global de almacenamiento que **posibilita a múltiples clientes compartir datos**. En estos sistemas cada fichero se almacena en un servidor, y el ancho de banda de acceso a un fichero se encuentra limitado por el acceso a un único servidor, lo que **convierte a los servidores en un cuello de botella en el sistema**. Este problema, conocido como la crisis de la E/S, sigue sin estar resuelto en sistemas distribuidos de propósito general, pero se han propuesto diferentes soluciones para resolver este problema, entre las que cabe destacar las siguientes:

- **Utilización de paralelismo** en el sistema de E/S con la distribución de los datos de un fichero entre diferentes dispositivos y servidores.
- Empleo de sistemas de **almacenamiento de altas prestaciones** (redes de almacenamiento).

3.1.1.1 Network File System

NFS es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local. Posibilita que **distintos sistemas conectados a una misma red accedan a ficheros remotos como si se tratara de locales**. Originalmente fue **desarrollado en 1984 por Sun Microsystems**, con el objetivo de que sea independiente de la máquina, el sistema operativo y el protocolo de transporte, esto fue posible gracias a que está implementado sobre los protocolos XDR (presentación) y ONC RPC (sesión). El protocolo NFS está incluido por defecto en los Sistemas Operativos UNIX y la mayoría de distribuciones Linux.

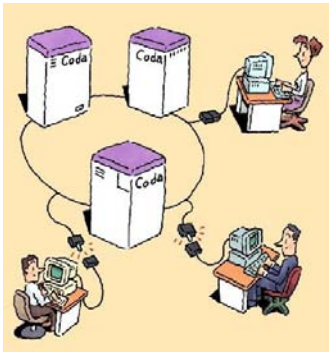
Sus principales características son:

- El sistema NFS está dividido al menos en **dos partes principales: un servidor y uno o más clientes.** Los clientes acceden de forma remota a los datos que se encuentran almacenados en el servidor.
- Las estaciones de trabajo locales utilizan menos espacio de disco debido a que los **datos se encuentran centralizados en un único lugar** pero pueden ser accedidos y modificados por varios usuarios, de tal forma que **no es necesario replicar la información**.
- Los usuarios no necesitan disponer de un directorio “home” en cada una de las máquinas de la organización. Los directorios “home” pueden crearse en el servidor de NFS para posteriormente poder acceder a ellos desde cualquier máquina a través de la infraestructura de red.
- También se pueden compartir a través de la red dispositivos de almacenamiento como disqueteras, CD-ROM y unidades ZIP. Esto puede reducir la inversión en dichos dispositivos y mejorar el aprovechamiento del hardware existente en la organización.

Todas **las operaciones sobre ficheros son síncronas**. Esto significa que la operación sólo retorna cuando el servidor ha completado todo el trabajo asociado para esa operación. En caso de una solicitud de escritura, el servidor escribirá

físicamente los datos en el disco, y si es necesario, actualizará la estructura de directorios, antes de devolver una respuesta al cliente. Esto garantiza la integridad de los ficheros.

3.1.1.2 CODA



El Sistema de Ficheros Distribuido Coda **es el sucesor de *Andrew File System* (AFS)** y es un desarrollo de la Universidad de Carnegie-Mellon como ejemplo de entorno de trabajo distribuido. Coda destaca sobre AFS por **permitir la Computación Móvil (trabajar en modo desconectado)**, soportar mejor la tolerancia a fallos del sistema (por ejemplo caída de los servidores o fallos de la red) y por disponer de técnicas de replicación de los servidores. Al ser gratuito, su código fuente está y está diseñado para **trabajar tanto en *LAN* como en *WAN***.

Tiene múltiples características que son deseables en la mayoría de sistemas de archivos. Además, tiene algunas características propias:

- **Puede funcionar sin conexión.**
- Es software libre.
- **Gran rendimiento gracias a la caché persistente en el cliente**.
- **Replicado de servidores**.
- Modelo de seguridad para **autenticación, cifrado y control de acceso.**
- **Funcionamiento continuado durante fallos de red.**
- **Ajuste del ancho de banda de red.**

Coda utiliza una caché local para proporcionar acceso a los datos del servidor cuando ocurren desconexiones en la red. Durante el funcionamiento normal, un usuario lee y escribe al sistema de archivos con normalidad, mientras el cliente obtiene todos los datos que ha marcado como importantes en el caso de una desconexión de red. Cuando se pierde la conexión de red, el cliente Coda sirve los datos desde su caché local y registra cualquier actualización. A este estado se le llama funcionamiento sin conexión. Al restablecerse la conexión, el cliente Coda pasa del funcionamiento sin conexión hacia un estado transitorio de "reintegración" donde las actualizaciones registradas se envían a los servidores. Cuando todas las actualizaciones se han reintegrado, el cliente vuelve al estado normal de funcionamiento con conexión.

Debido a sus características Coda tiene una serie de desventajas:

- Las operaciones de bloqueo de ficheros no están implementadas debido a que no es posible un algoritmo de bloqueo que tenga en cuenta un funcionamiento en modo desconectado.
- Existe un **problema de sincronización intrínseco al modo desconectado**: cuando al reconectar un cliente, un fichero ha cambiado tanto en el cliente como en el servidor, ¿cuál es la versión que se debe sincronizar con el resto del sistema?
- Existen diversos algoritmos, pero frecuentemente se requiere la mano del operador.
- La implementación de cuotas es limitada y sólo existe para los directorios (no existen cuotas para usuarios). Para solucionarlo se puede asignar un volumen por usuario, pero cambiar la cuota a un usuario es complicado porque los volúmenes Coda no son redimensionables.
- **Coda no es estable** y actualmente no se soportan bien volúmenes de más de 100 usuarios, ni mezcla de servidores Coda que no estén replicados (cada servidor Coda sirviendo un volumen independiente).
- Todas las operaciones de administración deben hacerse desde un cliente Coda sin que se pueda trabajar directamente con los volúmenes. Esto dificulta enormemente las tareas de mantenimiento y administración.
- **Una máquina no puede ser a la vez cliente y servidor Coda**.

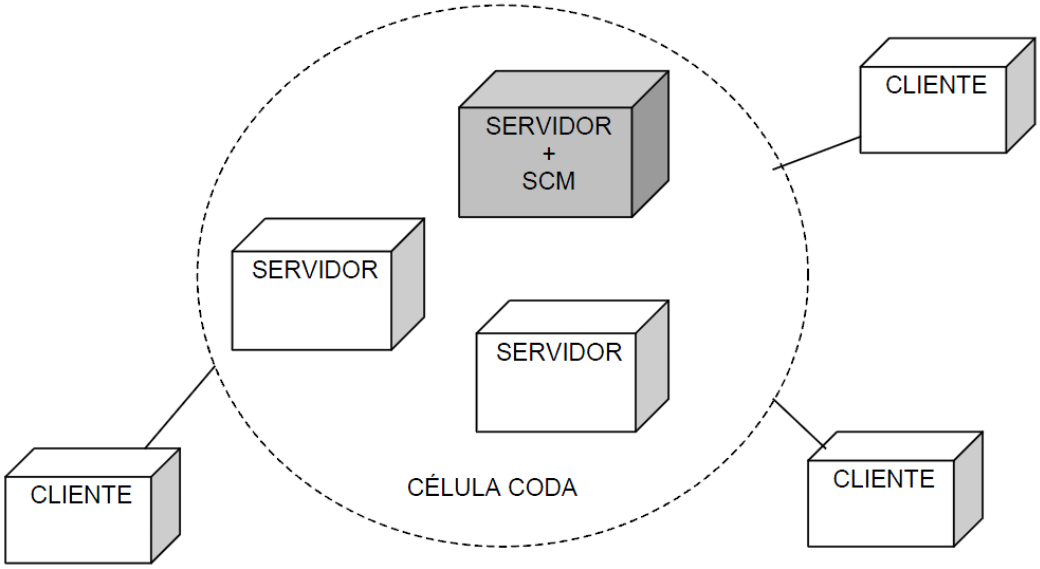


Figura 3. Arquitectura CODA

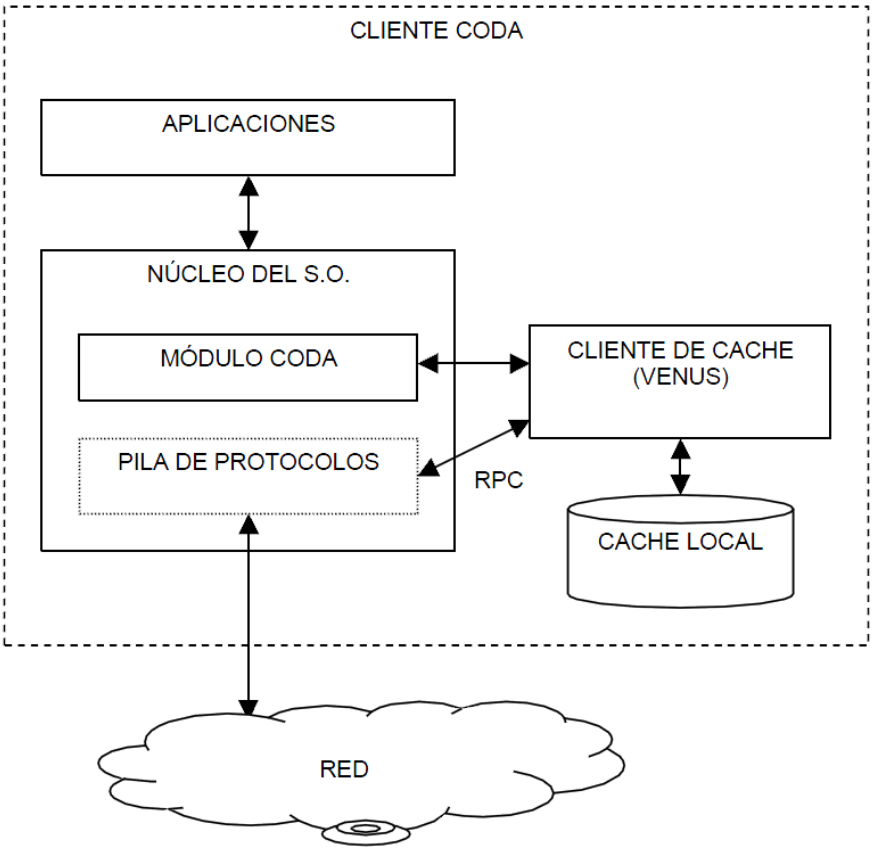


Figura 4. Cliente CODA

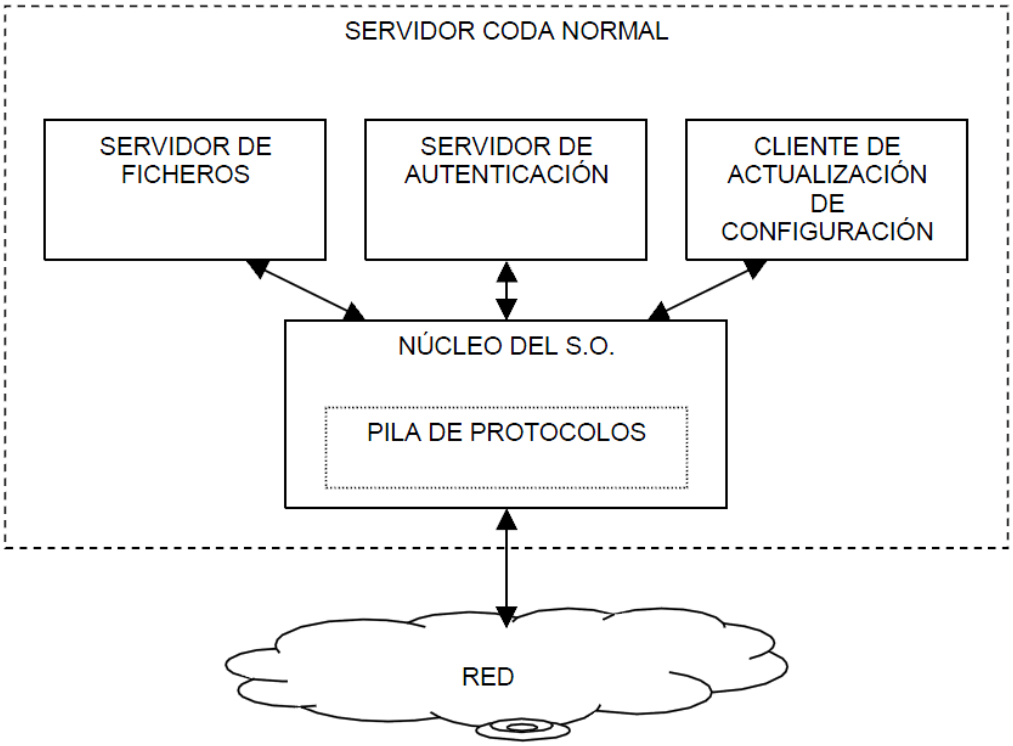


Figura 5. Servidor normal CODA

3.1.2 Sistema de ficheros paralelo

Un sistema de ficheros paralelo es aquél que **elimina el problema del cuello de botella de E/S, agregando de forma lógica múltiples dispositivos de almacenamiento independientes y nodos de E/S mediante un sistema de almacenamiento de alto rendimiento.**

En los sistemas de ficheros paralelos el ancho de banda puede incrementarse mediante:

- **Direccionamiento de disco independiente**, mediante el cual el sistema de ficheros puede acceder a datos de diferentes ficheros de forma concurrente.
- **Reparto de los datos en los nodos,** mediante el cual un solo fichero se puede acceder en paralelo.

Un sistema de ficheros paralelo opera, al igual que los sistemas de ficheros distribuidos, independientemente de las aplicaciones ofreciendo una mayor flexibilidad y generalidad que las bibliotecas.

3.1.2.1 General Parallel File System

GPFS (*General Parallel File System*) es un sistema de ficheros **desarrollado por IBM**. GPFS es particularmente apropiado en **entornos donde los sistemas distribuidos no ofrecen suficiente rendimiento de ancho de banda**. Siendo un entorno que permite a los usuarios compartir el acceso a los datos a través de *cluster*, posibilitando la interacción a través de las interfaces estándar de UNIX.

La fortaleza de GPFS se basa en:

- Mejora el rendimiento del sistema: permite que **múltiples nodos accedan simultáneamente a los datos** utilizando llamadas estándar del sistema, **balanceando la carga** y por lo tanto incrementando el ancho de banda disponible por cada nodo.
- Asegura la **consistencia de los datos**: utiliza un sofisticado sistema de administración que provee la consistencia de los datos mientras permite múltiple e independientes rutas para archivos con el mismo nombre.
- Alta recuperabilidad y disponibilidad de los datos: **crea registros *logs*** separados para cada uno de los nodos que intervienen en el sistema. Permite administrar el número de replicas que se desea manejar.

- **Alta flexibilidad del sistema**: los recursos no se encuentran congelados, se puede añadir o quitar discos al sistema mientras este se encuentra montado. Cuando la demanda es muy baja se puede reconfigurar la cargar del sistema a través de todos los discos configurados. También se puede **agregar nuevos nodos sin que el sistema sea detenido** y puesta en marcha nuevamente.
- Administración simplificada: los comandos de GPFS guardan la configuración en más de un archivo, conocido colectivamente como "*cluster* de datos". Los comandos de GPFS están diseñados para sincronizar los datos en cada uno de los nodos del sistema. De tal modo se asegura una exacta configuración de los datos.

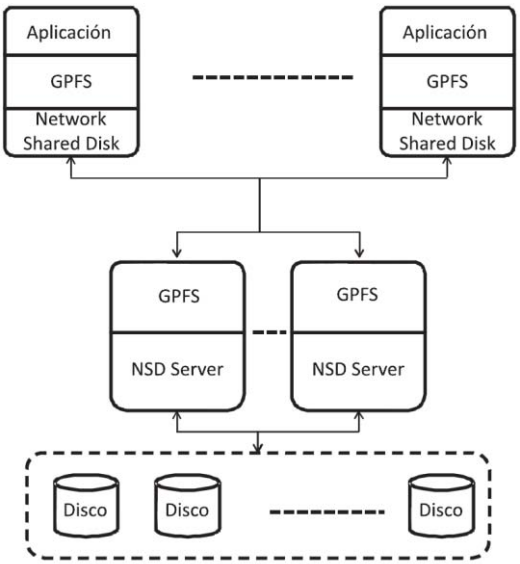


Figura 6. Arquitectura GPFS usando servidores de datos

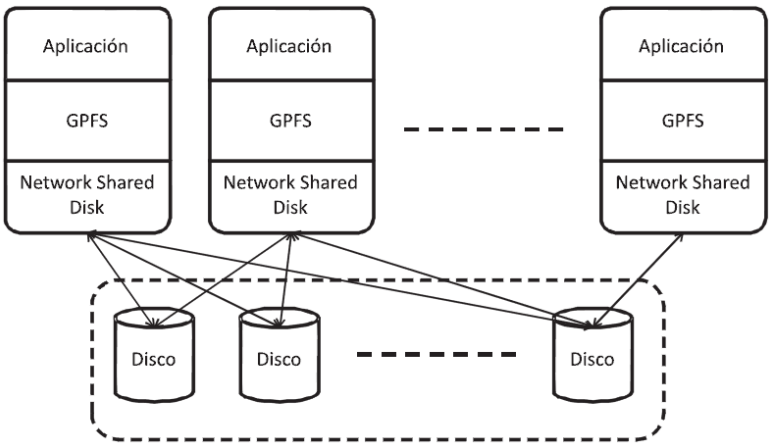


Figura 7. Arquitectura GPFS usando discos de forma directa

3.1.2.2 LUSTRE

LUSTRE es un sistema de ficheros paralelo **diseñado por Cluster File Systems**. En Lustre se considera a cada archivo almacenado en el sistema de archivos Lustre un objeto. Lustre presenta a todos los clientes una semántica POSIX estándar y **acceso concurrente lectura y escritura** para los objetos compartidos. Un sistema de archivos Lustre **tiene cuatro unidades funcionales**. Estas son: ***Meta data server* (MDS)** para almacenar los metadatos; un ***Object storage target* (OST)** para guardar los datos reales; un ***Object storage server* (OSS)** para manejar los OSTs; **cliente(s)** para acceder y utilizar los datos. Los OSTs son dispositivos de bloques. Un MDS, OSS, y un OST pueden residir en el mismo nodo o en nodos diferentes. Lustre no administra directamente los OSTs, y delega esta responsabilidad en los OSSs para asegurar la escalabilidad para grandes clusters y supercomputadores.

En un *Massively Parallell Processor* (MPP), los procesadores pueden acceder al sistema de archivos Lustre redirigiendo sus peticiones I/O hacia el nodo con el servicio lanzador de tareas si está configurado como un cliente Lustre. Aunque es el método más sencillo, en general proporciona un bajo rendimiento. Una manera ligeramente más complicada de proporcionar un rendimiento global muy bueno consiste en utilizar la biblioteca *liblustre*. *Liblustre* es una biblioteca de nivel de usuario que permite a los procesadores montar y utilizar el sistema de archivos Lustre como un cliente, sorteando la redirección hacia el nodo de servicio. Utilizando liblustre, los procesadores pueden acceder al sistema de archivos Lustre, incluso si el nodo de servicio en el que se lanzó el trabajo no es un cliente Lustre. Liblustre proporciona un mecanismo para mover datos directamente entre el espacio de aplicación y los OSSs de Lustre sin necesidad de realizar una copia de datos a través del núcleo ligero, logrando así una baja latencia, y gran ancho de banda en el acceso directo de los procesadores al sistema de archivos Lustre.

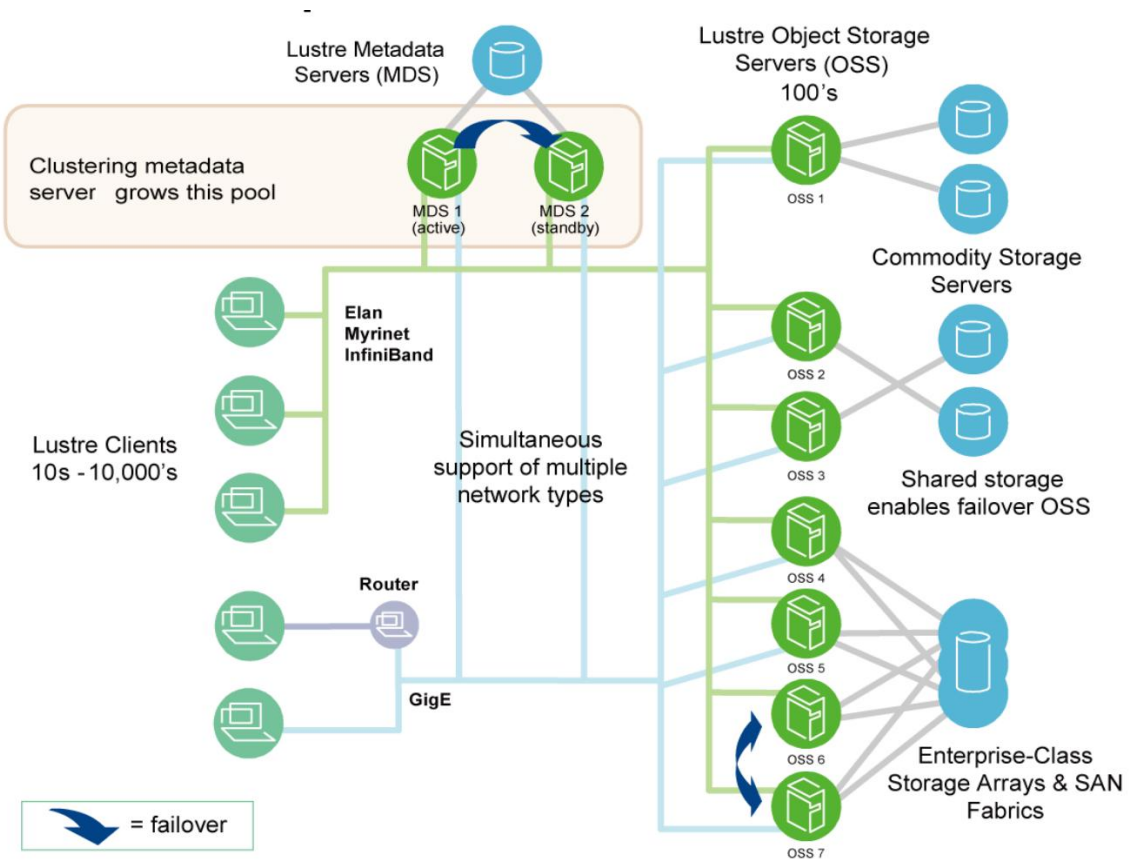


Figura 8. Arquitectura LUSTRE

3.1.2.3 *Parallel Virtual File System*

PVFS *(Parallel Virtual File System*) es un sistema de ficheros paralelo, que permite a las aplicaciones, indistintamente de si estas se ejecutan en forma paralela o secuencia, almacenar y **acceder a ficheros cuyos datos se encuentran distribuidos a través de un conjunto de servidores de E/S**. PVFS ha sido **desarrollado para *clusters* Linux**, proporcionando un gran ancho de banda en **operaciones de lectura y escritura concurrentes** realizadas desde múltiples procesos ligeros a un fichero común. Los datos de los ficheros se distribuyen a través de los discos de las máquinas del *clusters* Linux y proporcionando la apariencia de un único sistema de ficheros Unix.

En 2001 arrancó el proyecto PVFS2, en el que se ha reescrito el código y se ha modificado el núcleo de la versión 1 de PVFS para que sea más apropiado para el entorno en el que los sistemas de ficheros paralelos son desplegados. De esta forma se ha conseguido un sistema de ficheros paralelos más adecuado a las necesidades, robusto y de alto rendimiento.

Las características que posee la versión 2 de PVFS son:

- Distribución de datos flexible. PVFS1 utiliza el **patrón de distribución round-robin** para realizar la distribución de datos de los ficheros entre los servidores. PVFS2 no solo permite el uso del patrón de distribución *Round-Robin* sino que **incluye un sistema modular para añadir nuevos patrones de distribución** en el sistema y el uso de ello para los nuevos ficheros.
- Información de metadatos distribuida. PVFS1 contemplaba un único servidor de metadatos, pero se convierte en un único punto de fallos, puesto que los datos no se encuentran replicados, y es un cuello de botella. En PVFS2 **los metadatos se encuentran distribuidos en un conjunto de servidores**, que pueden coincidir con los servidores de datos.
- Proporciona múltiples interfaces, incluso un interfaz de MPI4IO vía ROMIO y otra para acceder al sistema de ficheros tradicional mediante el uso de un módulo del kernel de Linux.
- Utiliza **servidores y clientes sin estado**, lo cual, facilita la recuperación de fallos en el sistema.
- **Soporte de concurrencia** explícito. En PVFS2 los hilos o procesos ligeros son usados cuando son necesarios para proporcionar acceso no bloqueante a todos los dispositivos y evitar la serialización de las operaciones independientes.
- Soporte de redundancia de datos y metadatos. PVFS1 no soporta redundancia de datos. En PVFS2 utiliza el **enfoque RAID para proporcionar tolerancia de fallos de disco**, pero si un servidor desaparece, los datos que contenía son inaccesibles hasta que el servidor se recupera.
- **Soporta clusters heterogéneos**.
- Presenta un diseño modular.

En cuanto a los sistemas de ficheros paralelos, **su principal problema es que están especialmente pensados para máquinas paralelas** y no se integran adecuadamente en entornos distribuidos de propósito general. Además, cada sistema de ficheros paralelo utiliza una estructura de fichero paralelo diferente, incompatible con la de otros sistemas.

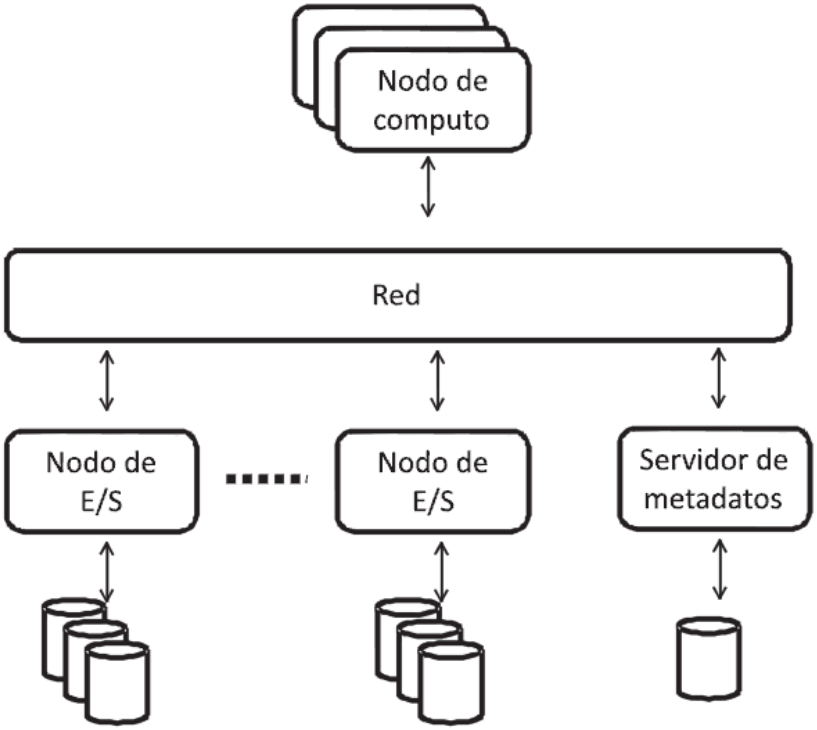


Figura 9. PVFS versión 1

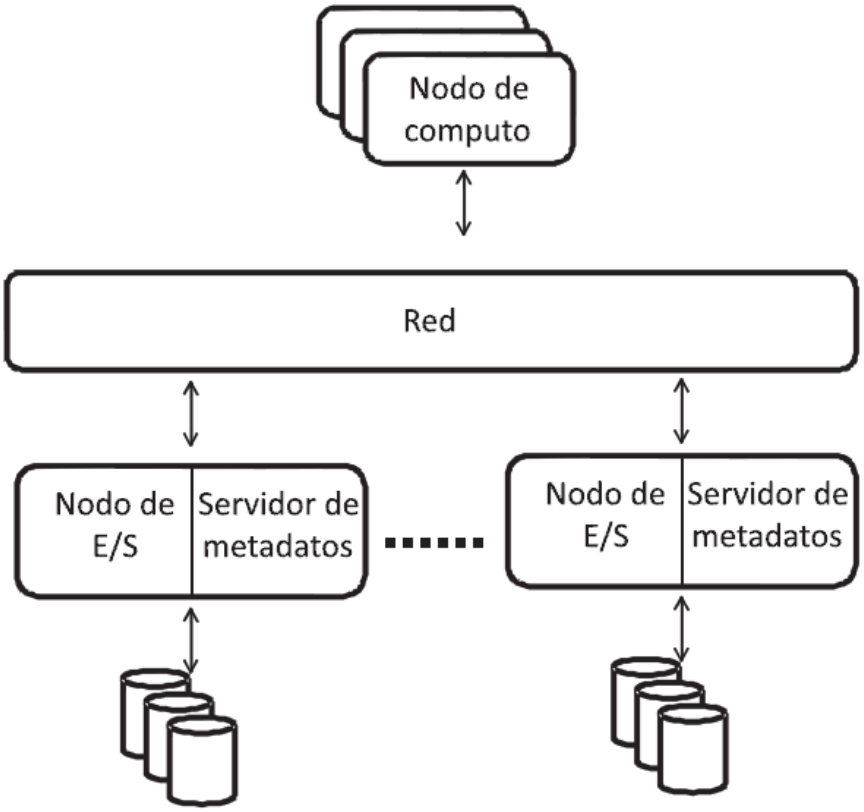


Figura 10. PVFS versión 2

3.1.3 Google File System

El sistema de ficheros Google (GFS) se ha diseñado e implementado para satisfacer la creciente demanda de procesamiento de datos de Google. GFS comparte algunos de los objetivos de los sistemas de archivos tradicionales; tales como **rendimiento, escalabilidad, fiabilidad y disponibilidad**. Su diseño ha estado dirigido por observaciones de la carga de trabajo y entorno tecnológico de Google.

Suposiciones del diseño

- El sistema está formado por muchos componentes que fallan con frecuencia.
- El sistema almacena un número pequeño de grandes ficheros.
- La carga de trabajo principalmente consta de dos tipos de lecturas: lecturas de grandes flujos de datos y lecturas pequeñas aleatorias.
- La carga de trabajo también incluye muchas escrituras secuenciales y operaciones que añaden datos a ficheros.
- El sistema debe implementar eficientemente un mecanismo para que múltiples clientes puedan añadir concurrentemente datos a un mismo fichero.
- Un ancho de banda grande es más importante que una latencia baja.

Interfaz. La API

GFS proporciona una interfaz para el sistema de archivos, aunque **no implementa un API estándar tal como POSIX**. **Las operaciones más comunes permitirán crear, borrar, abrir, borrar, leer y escribir ficheros**. Además, GFS tiene operaciones de duplicación y de adición de información a ficheros.

La arquitectura del sistema

- Un cluster GFS consta de **un único maestro** y **varios servidores de bloques** a los que acceden múltiples **clientes**.
- Los **ficheros están divididos en bloques de un tamaño fijo**. Por fiabilidad, cada bloque esta **replicado en varios servidores** de bloques. Por defecto, se almacenan tres replicas.

- El servidor **maestro mantiene metadatos del sistema de ficheros** y controla actividades del sistema.
- El código cliente GFS insertado en cada aplicación utiliza la API del sistema de archivos y se comunica con el servidor maestro y servidores de bloques para leer o escribir datos utilizados en la aplicación.
- **Ni los clientes ni los servidores de bloques mantienen los ficheros de datos en la cache.**

Un solo servidor maestro

Tener **un único servidor maestro simplifica enormemente el diseño** y posibilita al maestro hacer sofisticados emplazamientos de bloques y decisiones de replicación utilizando información global.

A continuación se explican las interacciones que hay que realizar en una operación de lectura:

1. El cliente traduce el nombre del fichero y el desplazamiento de bytes especificado por la aplicación en un índice de bloque dentro del archivo.
2. Envía al maestro una petición que contiene el nombre del archivo y el índice del bloque.
3. El maestro responde con el correspondiente identificador de bloque y la localización de la réplica.
4. El cliente almacena en la caché esta información usando el nombre del archivo y el índice de bloque como clave.
5. El cliente envía una petición a una de las réplicas, normalmente a la más cercana.

El tamaño de bloque

El **tamaño de bloque es uno de los parámetros de diseño clave**. Google utiliza un tamaño de bloque de 64 Mb. Un tamaño de bloque grande ofrece importantes ventajas:

1. **Reduce la necesidad de interactuar con el maestro** porque las lecturas y escrituras realizadas en un mismo bloque requieren solo una única petición al maestro para obtener la información de localización del bloque.
2. En un bloque grande, es más probable que el cliente ejecute varias operaciones, así que se puede reducir las conexiones manteniendo una

conexión TCP persistente con el servidor de bloques durante un amplio periodo de tiempo.

3. **Se reduce el tamaño de los metadatos** almacenados en el servidor maestro.

Por otro lado, un tamaño de bloque grande puede tener desventajas:

1. Un fichero pequeño tiene un pequeño número de bloques, quizás solo uno. El servidor de bloques que almacena estos bloques podría convertirse en un punto crítico si varios clientes están accediendo a un mismo archivo. Este problema se puede arreglar almacenando tales ficheros con un mayor factor de replicación. Otra solución potencial es permitir a los clientes leer datos de otros clientes en tales situaciones.

Los Metadatos

El servidor maestro almacena **tres tipos de metadatos: el espacio de nombres de los bloques y ficheros; el mapeo de los ficheros a los bloques; y la localización de cada réplica de un bloque.**

Los dos primeros tipos se mantienen persistentes registrando los cambios mediante una operación log. Por el contrario, no se almacena la información de localización de los bloques persistentemente. Todos los metadatos se almacenan en memoria para que las operaciones sean rápidas.

Una restricción fundamental es que el número de bloques y por lo tanto la **capacidad del sistema completo está limitada por el cantidad de memoria que tiene el maestro.**

El coste de añadir memoria extra al maestro es un pequeño precio a pagar por la simplificación, fiabilidad, rendimiento y flexibilidad que se gana almacenando metadatos en memoria. Inicialmente se intentó mantener la información de localización de los bloques de manera persistente en el maestro, pero se decidió que era más simple pedir los datos a los servidores de bloque al iniciar la conexión y posteriormente hacerlo de forma periódica.

La operación log contiene un registro histórico de los cambios críticos en los metadatos. Puesto que el registro histórico de la operación log es crítico, se debe almacenar con fiabilidad, por lo que se replica en varias máquinas remotas. El maestro recupera el estado del fichero de archivos repitiendo los cambios registrados en el log.

El maestro crea puntos de control de sus estados cada vez que el log crece más allá de un cierto tamaño, así que se puede recuperar cargando el punto de control desde el disco local y repetir solo un número limitado de registros log a partir de ese tamaño.

Modelo de consistencia

GFS tiene un modelo de consistencia flexible que **soporta aplicaciones altamente distribuidas** y es relativamente simple y eficiente de implementar. Los **cambios del espacio de nombres de un fichero son atómicos**.

El estado de una región de un archivo después de un cambio de datos depende del tipo de cambio, si tiene éxito o falla, y si hay cambios concurrentes. Una región puede estar en un estado definido o indefinido.

Después de una secuencia de cambios con éxito, la región del fichero cambiado se garantiza que sea definida y contenga los datos escritos por el último cambio. GFS consigue esto:

1. **Aplicando cambios a un bloque en el mismo orden en todas sus réplicas**.
2. Usando **números de versiones de bloques para detectar alguna réplica que es antigua** porque no ha realizado un cambio porque su servidor de bloques estaba caído.

Después de un cambio con éxito, los fallos de los componentes pueden corromper o destruir datos.

Actualización y orden de los cambios / mutación

Un cambio es una operación que modifica los datos o los metadatos de un bloque, como por ejemplo escribir o añadir.

Se usa la actualización para mantener una consistencia en el orden de cambio a lo largo de las réplicas.

El proceso de actualización se realiza siguiendo estos pasos:

1. Los clientes preguntan al maestro que servidor de bloques almacena la actual actualización para el bloque y la localización de otras réplicas.

2. El maestro responde con la identidad del principal y la localización de otras réplicas (secundario).
3. El cliente envía los datos a todas las réplicas.
4. Todas las réplicas reconocen los datos recibidos y el cliente envía una petición de escritura al principal.
5. El principal remite la petición de escritura a todas las replicas secundarias.
6. Todos los secundarios responde al principal indicando que han completado la operación.
7. El principal responde al cliente.

Si una escritura de una aplicación es extensa o incluye un límite de bloque, el código del cliente GFS se descompone en varias operaciones de escritura.

Flujo de datos

El flujo de datos está desacoplado del flujo de control para usar la red eficientemente. Mientras el control fluye desde el cliente al principal y luego a todos los secundarios, los datos son colocados en línea a lo largo de una cadena por un servidor de bloques en forma de tubería.

El objetivo es usar completamente el ancho de banda de la red de máquinas, evitando cuellos de botella en la red y enlaces de alta latencia, y minimizando la latencia para colocar los datos.

Añadir información a ficheros

GFS proporciona una operación atómica para añadir información a ficheros. La operación de añadir información es un tipo de mutación y sigue el flujo de control de explicado en el apartado 1 (Actualización y orden de los cambios / mutación) con solo una pequeña lógica extra del principal:

- El cliente pone los datos en todos los últimos bloques del fichero.
- Luego, se envía las peticiones al principal.
- El principal verifica para ver si la operación en el bloque actual causaría que el bloque excediera el tamaño máximo (64 MB). Si es así, se rellena el bloque hasta el tamaño máximo, se le dice a los secundarios que hagan lo mismo, y se contesta al cliente indicando que la operación debería volverse intentar en el próximo bloque.

- Si la operación es adecuada para el tamaño máximo, el principal añade los datos a sus replicas, le dice a los secundarios que escriban los datos en el desplazamiento exacto, y finalmente responde exitosamente al cliente.
- Si la operación falla en alguna de las réplicas, el cliente reintenta la operación.

GFS no garantiza que todas las réplicas sean idénticas, solo garantiza que los datos se escriban como una unidad atómica.

Duplicación

La operación de duplicación hace una copia de un fichero o un árbol de directorios.

Los usuarios lo usan para crear copias secundarias de grandes conjuntos de datos, o para crear puntos de control del estado actual antes de realizar cambios. Cuando el maestro recibe una petición de duplicación:

- Primero elimina cualquier actualización pendiente sobre los bloque del fichero que se desea duplicar.
- Después que las actualizaciones han sido rechazadas o han expirado, el maestro registra la operación en disco.
- Finalmente duplica los metadatos de los ficheros fuente y el árbol de directorios. El nuevo fichero duplicado apuntará al mismo bloque que el archivo fuente.

Gestión del espacio de nombres y bloqueo

GFS representa lógicamente su espacio de nombres como una tabla que mapea los caminos de nombres completos a un metadato. Cada operación del maestro realiza un conjunto de bloqueos antes de su ejecución. Por ejemplo para realizar una operación sobre el fichero /d1/d2/……../dn/leaf hay que realizar una serie de bloqueos para evitar conflictos.

Una propiedad importante de este esquema de bloqueo es que se permite mutaciones concurrentes en el mismo directorio.

Los bloqueos son activados en un orden total consistente para prevenir llegar a un punto muerto: estos primero se ordenan por nivel en el árbol de espacio de nombres y lexicalmente dentro del mismo nivel.

Emplazamiento de réplicas

Un clúster GFS está fuertemente distribuido en más de un nivel. Normalmente **tiene cientos de servidores de bloques desplegados a lo largo de varias subredes** de máquinas. La distribución multinivel presenta un gran desafió para distribuir datos proporcionando escalabilidad, fiabilidad y disponibilidad.

La política **de emplazamiento de réplicas** de bloques tiene dos propósitos: **maximizar la fiabilidad y disponibilidad de los datos y maximizar la utilización del ancho de banda de la red.** Se debe desplegar también réplicas a lo largo de las subredes para asegurar que algunas réplicas del bloque permanezcan disponibles incluso si una subred entera se daña o se desconecta.

Creación, replicación y rebalanceo

Cuando el maestro crea un bloque, este elige donde colocar las réplicas vacías inicialmente. Se consideran varios factores:

- Las réplicas nuevas se pueden colocar en servidores de bloques donde la utilización del espacio del disco sea menor.
- Se debería limitar el número de creaciones recientes sobre cada servidor de bloques.
- Como se dijo anteriormente, se desea desplegar réplicas de bloques a lo largo de las subredes.

El maestro replica un bloque en cuanto el número de réplicas disponibles está por debajo de un umbral especificado por el usuario.

Cada bloque que necesite ser replicado se le asigna una prioridad basada en varios factores.

El maestro elige el bloque con mayor prioridad y se “clona” ordenando a alguno servidor de bloques que copie los datos del bloque directamente de una réplica válida existente. La réplica válida se ubica siguiendo objetivos similares a los de creación. El maestro balancea periódicamente las réplicas: examina la distribución de réplicas actual y mueve las réplicas para mejorar el espacio del disco y balancear la carga. El criterio de emplazamiento de la nueva réplica es similar a lo discutido anteriormente. El maestro debe también elegir que réplica existente borrar.

Recogida de basura

Después de que un fichero es borrado, GFS no recupera la disponibilidad de almacenamiento físico.

El fichero es renombrado con un nombre oculto que incluye la etiqueta de borrado.

Durante un escaneo regular del maestro del espacio de nombre del sistema de ficheros, se borra cualquier fichero oculto que fue borrado hace más de tres días. Cuando el fichero oculto es eliminado del espacio de nombres, sus metadatos de memoria son borrados.

En un escaneo regular similar del espacio de nombres de los bloques, el maestro identifica los bloques “huérfanos” y borra los metadatos de estos bloques. La recogida de basura con recuperación de almacenamiento ofrece varias ventajas sobre el borrado instantáneo:

1. Es simple y fiable en un sistema distribuido a gran escala donde los fallos de los componentes son comunes.
2. El escaneo del espacio de nombres y la conexión con los servidores de bloques se realiza en segundo plano.
3. El retardo en la recuperación del espacio proporciona seguridad contra accidentes o borrados irreversibles.

La principal desventaja es que el retardo a veces dificulta el esfuerzo del usuario para aprovechar el uso cuando el almacenamiento es ajustado. Los usuarios tienen permitido aplicar diferentes políticas de recuperación y replicación en distintas partes del espacio de nombres.

Borrado de réplicas antiguas

Las réplicas de bloques pueden volverse antiguas si un servidor de bloques falla y pierde mutaciones de un bloque mientras está caído. Para cada bloque, el maestro mantiene un número de versión de bloque para distinguir entre las réplicas actuales y las antiguas.

Cuando el maestro otorga una actualización de un bloque, este incremente su número de versión de bloque e informa a las réplicas actuales. El maestro detectará que este servidor de bloques tiene una réplica antigua cuando el servidor de bloque se

reinicie y le informe de su subconjunto de bloques y sus números de versión asociados. El maestro elimina las réplicas antiguas en la recogida de basura.

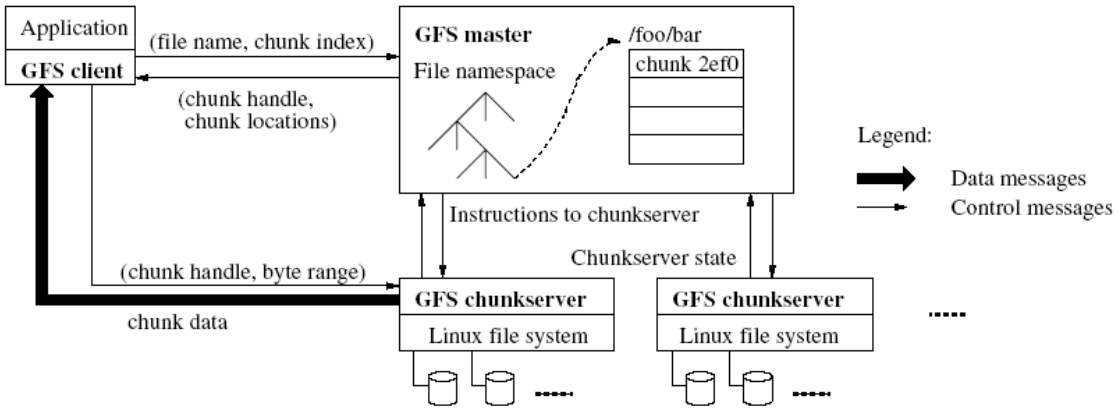


Figura 11. Google File System

3.2 *Filesystem in Userspace*

FUSE es un módulo cargable en el núcleo para sistemas operativos tipo Unix, que **permite a usuarios no privilegiados crear sus propios sistemas de archivos sin necesidad de editar el código del núcleo**. Esto se logra mediante la ejecución del código del sistema de archivos en el espacio de usuario, mientras que el módulo FUSE sólo proporciona un "puente" a la interfaz del núcleo real. FUSE fue oficialmente integrado en el *kernel* de Linux en la versión 2.6.14.

FUSE es realmente útil **para la creación de sistemas de archivos virtuales**. A diferencia de los tradicionales sistemas de archivos, que, en esencia, guardan y recuperan los datos desde un disco, los sistemas de archivos virtuales en realidad no almacenan datos propios. Actúan como una visualización o traducción de un sistema de archivos existente o dispositivo de almacenamiento.

Cuando se monta un programa FUSE sobre un directorio todas las llamadas que se realicen sobre este directorio son derivadas a este programa, que es el encargado de devolver el valor de la llamada.

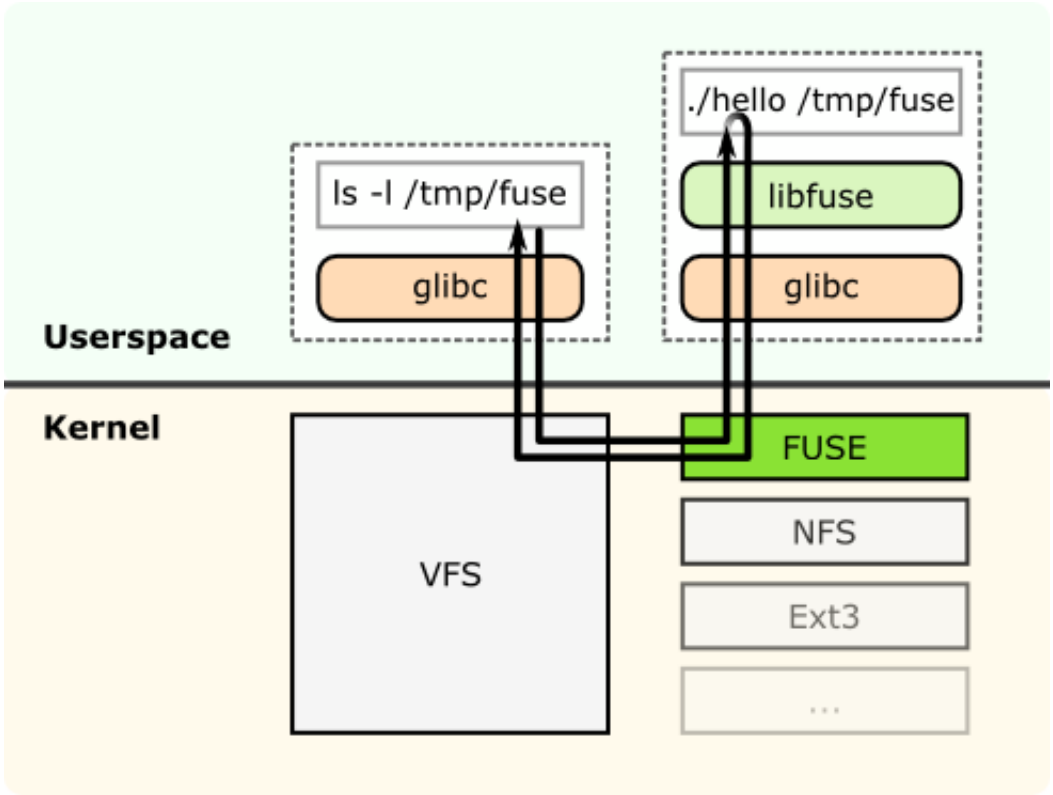


Figura 12. FUSE

3.2.1 Proyectos realizados con FUSE

Lista de proyectos divididos en categorías (solo algunos ejemplos):

- **ArchiveFileSystems:** acceder a los archivos dentro de archivos (tar, cpio, zip, etc.)
    - **fuse.gunzip:** acceder a ficheros comprimidos con gzip.
    - **Rar2fs:** permite montar un archivo rar, o un directorio que contenga numerosos archivos rar en modo solo lectura.
- **CompressedFileSystems:** - acceso a los archives comprimidos en una imagen (gz, zlib, LiveCDs, etc.)
    - **Fusecram:** permite montar imágenes caramfs.
    - **compFUSEd:** permite montar en modo lectura y escritura imágenes de tipo .lzo .zlib y .bzip2.
- **DatabaseFileSystems:** guardar archives dentro de una base de datos relacional (MySQL, BerkeleyDB, etc.) o permitir búsquedas usando tags o consultas SQL.
    - **LAFS:** sistema que permite organizar y clasificar ficheros y directorios con metadatos almacenados en una base de datos PostgreSQL.
    - **Mysqlfs:** almacena los ficheros en una base de datos mysql.
- **EncryptedFileSystems:** almacenamiento de archives de una forma más segura mediante el uso de una clave secreta.
    - **PhoneBook:** agenda que dependiendo de la clave muestra una información u otra.
    - **MinorFs:** permite el almacenamiento privado de procesos pseudo persistentes. Esto permite que los programas que se ejecutan por un usuario puedan mantener algunos datos a salvo de todo malware que pudiese estar ejecutándose.
- **MediaFileSystems:** almacenamiento de archives en dispositivos multimedia como cámaras y reproductores de música o accede y clasificar archives multimedia.
    - **SIEF:** permite acceder a los archivos almacenados en la memoria de teléfonos Siemens.
    - **FUSEPod:** permite acceder a un iPod.
- **HardwareFileSystems:** facilitar el acceso a hardware extraño u obsoleto.
    - **OWFS:** (One Weird File System) permite ver todos los Dallas 1-Wire sensores, iButtons y chips de memoria como un sistema de archivos.
    - **k8055fs:** permite controlar la interfaz USB Velleman K8055.

- **MonitoringFileSystems:** facilitar log de operaciones.
    - **LoggedFS:** es un sistema de archivos que permite ver todas las operaciones individuales que ocurre en un sistema de archivos. Se puede elegir el tipo de archivos que se desea registrar. Entonces usted puede ver leer lo que sucede en estos archivos.
    - **AMQNotifyFS:** envía una notificación a traves de un BUS ActiveMQ cuando se modifica un archivo.
- **NetworkFileSystems:** almacenar archives en equipos remotos como servidores de archives y sitios web.
    - **Sshfs:** Este es un cliente del sistema de archivos basado en el protocolo de transferencia de archivos SSH. Como la mayoría de los servidores SSH ya son compatibles con este protocolo es muy fácil de configurar: es decir, del lado del servidor no hay nada que hacer. Por el lado del cliente el montaje del sistema de archivos es tan fácil como conectarse al servidor con ssh.
    - **Httpfs:** monta cualquier archivo accesible mendiante http, en modo lectura, el servidor debe soporta HTTP/1.1.
- **NonNativeFileSystems:** accede a sistemas de ficheros que no son un estándar de Linux (NTFS, ZFS, etc.)
    - **ZFS:** permite acceder al sistema de ficheros ZFS, diseñado originalmente por Sun Microsystems para el sitema operativo OpenSoralis.
    - **FUR:** permite montar el sistema de ficheros Windows CE (modo lectura y escritura) y registro solo en modo lectura.
- **UnionFileSystems:** unir diversos sistemas de ficheros en un solo árbol.
    - **FunionFS:** combina 2 puntos de montaje, uno de solo lectura y otro de lectura-escritura. Los datos se leen del punto de solo lectura si no están presentes en el montaje de lectura-escritura.
    - **Mhddfs:** combina varios puntos de montaje en uno solo.
- **VersioningFileSystems:** sistema de ficheros con histórico de versiones anteriores y proporcionan acceso as sistemas de control de versiones.
    - **CopyFS:** cuando se modifica un archivo cualquier versiona anterior se mantiene, pudiendo volver a una versión anterior cuando se quiera.
    - **SvnFs:** permite acceder a los repositorios Subversion a través de un punto de montaje FUSE.

Dentro de los proyectos que hacen uso de una base de datos se pueden dividir en 3 grandes grupos:

- Uso de la base de datos para el almacenamiento de metadatos. Como por ejemplo el proyecto **Tagsistant** que asocia etiquetas a los ficheros actuales del sistema.
- Uso de la base de datos para el almacenamiento puro de datos. Que sería el caso del sistema de ficheros a desarrollar en este Proyecto Fin de Carrera.
- Uso mixto de la base de datos. Como por ejemplo el proyecto **CopyFS** que almacena los datos de los ficheros y a su vez metadatos del tipo fecha y hora de creación y modificación.

3.3 BerkeleyDB

Es una base de datos incrustada con **API para C, C++, Java, Perl, Python, Ruby, Tcl y muchos otros lenguajes de programación.** Soporta múltiples datos para una misma clave. Berkeley DB permite miles de hilos de control manipulando bases de datos de hasta 256 terabytes en muchos sistemas, incluidos la mayoría de los basados en UNIX y Windows, e incluso sistema operativos de tiempo real.

Una de las características más atractivas que ofrece es que **permite almacenar la base de datos completamente en memoria**, aumentando con ello la velocidad de las lecturas y escrituras.

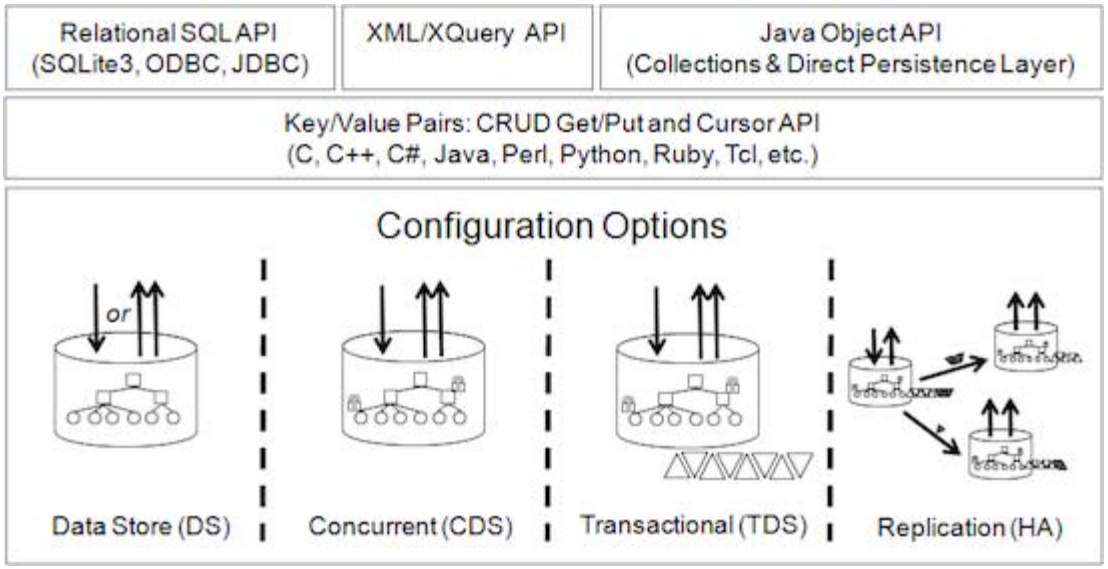


Figura 13. BerkeleyDb

3.4 Comunicación

En este apartado se trataran algunos protocolos de comunicación, como Sockets, Remote Procedure Call, Xml Remote Procedure Call y Web Service.

3.4.1 Sockets

Socket es un método para la comunicación entre un programa del cliente y un programa del servidor en una red, **permitiendo intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada**.

Los *sockets* permiten implementar una **arquitectura cliente-servidor o *peer-to-peer***. La comunicación debe ser iniciada por uno de los programas que se denomina programa "cliente". El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa "servidor".

Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

- Que **un programa sea capaz de localizar al otro**.
- Que ambos programas sean **capaces de intercambiarse cualquier secuencia de octetos**, es decir, datos relevantes a su finalidad.

Para ello son necesarios los tres recursos que originan el concepto de socket:

- **Un protocolo de comunicaciones**, que permite el intercambio de octetos.
- **Un par de direcciones del protocolo de red** (dirección IP, si se utiliza el protocolo TCP/IP), que identifican la computadora de origen y la remota.
- **Un par de números de puerto**, que identifican a un programa dentro de cada computadora.

Las propiedades de un socket dependen de las características del protocolo en el que se implementan. El protocolo más utilizado es *Transmission Control Protocol*; una alternativa común a éste es *User Datagram Protocol*.

Cuando se implementan **con el protocolo TCP**, los sockets tienen las siguientes propiedades:

- **Son orientados a la conexión**.
- Se **garantiza la transmisión** de todos los octetos sin errores ni omisiones.
- Se garantiza que **todo octeto llegará a su destino en el mismo orden en que se ha transmitido.**

Estas propiedades son muy importantes para garantizar la corrección de los programas que tratan la información.

El protocolo UDP es un protocolo no orientado a la conexión. Sólo se garantiza que si un mensaje llega, llegue bien. En ningún caso se garantiza que llegue o que lleguen todos los mensajes en el mismo orden que se mandaron. Esto lo hace adecuado para el envío de mensajes frecuentes pero no demasiado importantes, como por ejemplo, mensajes para los refrescos (actualizaciones) de un gráfico.

3.4.2 Remote Procedure Call

RPC es un protocolo que **permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos**. El protocolo es un gran avance sobre los sockets usados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando éstas encapsuladas dentro de las RPC.

Las RPC son **muy utilizadas dentro del paradigma cliente-servidor**. Siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando éste de vuelta el resultado de dicha operación al cliente.

Hay distintos tipos de RPC, muchos de ellos estandarizados como pueden ser el RPC de Sun denominado ONC RPC (RFC 1057), el RPC de OSF denominado DCE/RPC y el Modelo de Objetos de Componentes Distribuidos de Microsoft DCOM, aunque ninguno de estos es compatible entre sí. La mayoría de ellos utilizan un lenguaje de descripción de interfaz (IDL) que define los métodos exportados por el servidor.

3.4.3 Xmlrpc

XML-RPC es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes.

Es un protocolo muy simple ya que solo define unos cuantos tipos de datos y comandos útiles, además de una descripción completa de corta extensión. La simplicidad del XML-RPC está en contraste con la mayoría de protocolos RPC que tiene una documentación extensa y requiere considerable soporte de software para su uso.

Fue **creado por Dave Winer** de la empresa UserLand Software en asociación con Microsoft en el año 1998. Al considerar Microsoft que era muy simple decidió añadirle funcionalidades, tras las cuales, después de varias etapas de desarrollo, el estándar dejó de ser sencillo y se convirtió en lo que es actualmente conocido como SOAP. Una diferencia fundamental es que en los procedimientos en SOAP los parámetros tienen nombre y no interesan su orden, no siendo así en XML-RPC.

Los principales tipos de datos son: array, base64, boolean, data/time, doublé, integer, string, struct, nil.

3.4.4 Web service

Un servicio web es un protocolo de comunicación que sirve para intercambiar datos entre aplicaciones. Distintas aplicaciones software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos.

La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

Ventajas:

- Aportan interoperabilidad entre aplicaciones software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

Desventajas:

- Su rendimiento es bajo si se compara con otros modelos de comunicación.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

4. Análisis y Diseño

En este apartado se detallaran las características y funcionalidades del sistema, así como diversos diagramas que ayudaran a comprender mejor el diseño del sistema.

4.1 Características y funcionalidades del sistema

- En el sistema existirán **nodos de 3 tipos: Cliente, Metadatos e IOP**.
- **Los nodos Cliente estarán comunicados con el nodo de Metadatos y con los nodos IOP.**
- **Los nodos IOP estarán comunicados entre sí y con el nodo de Metadatos.**

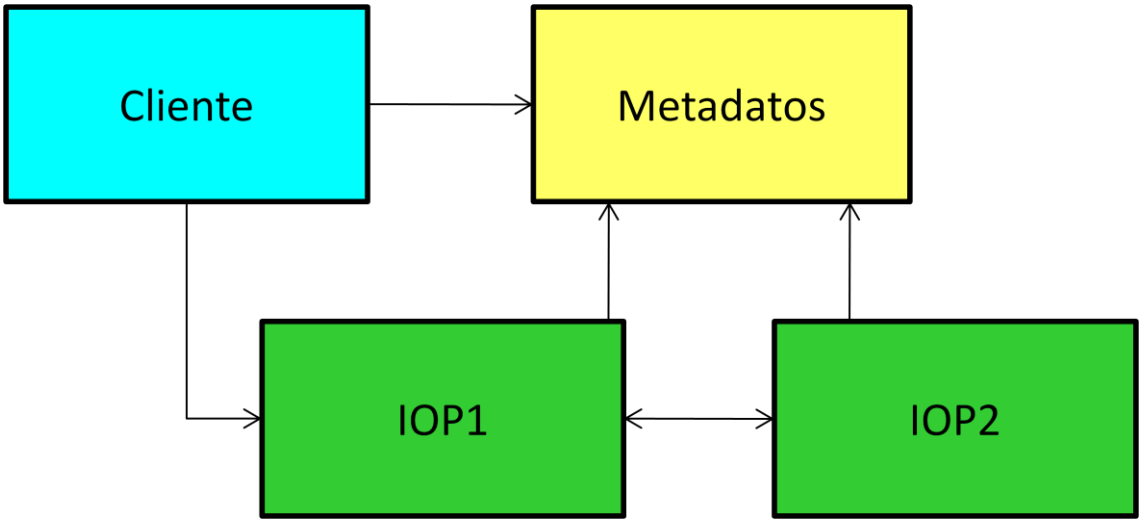


Figura 14. Nodos y comunicación

- **Nodo de Metadatos**:
  - Se considera que esta **siempre activo**.
  - Almacena la red a la que pertenecen los nodos IOP y su dirección IP en una base de datos.
  - Se divide en **3 capas**: servidor RPC, lógica del programa y base de datos.

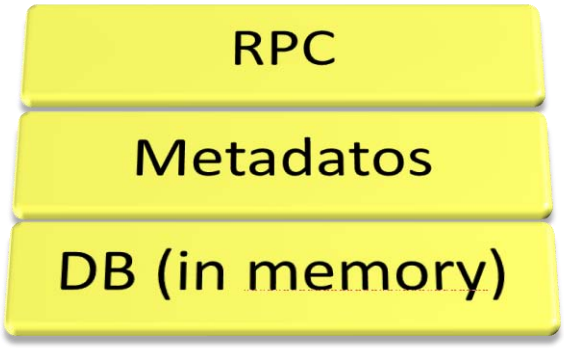


Figura 15. Capas nodo de Metadatos

  - Llamadas RPC del nodo de Metadatos (figura):
    - **Obtener nodos IOP de una red** (O): devuelve la lista de direcciones IP de los nodos IOP pertenecientes a una red de IOP.
    - **Registrar nodo IOP en una red** (R): añade la dirección IP de un nodo IOP a una red y devuelve la lista de direcciones IP de los nodos que ya estaban registrados en dicha red.

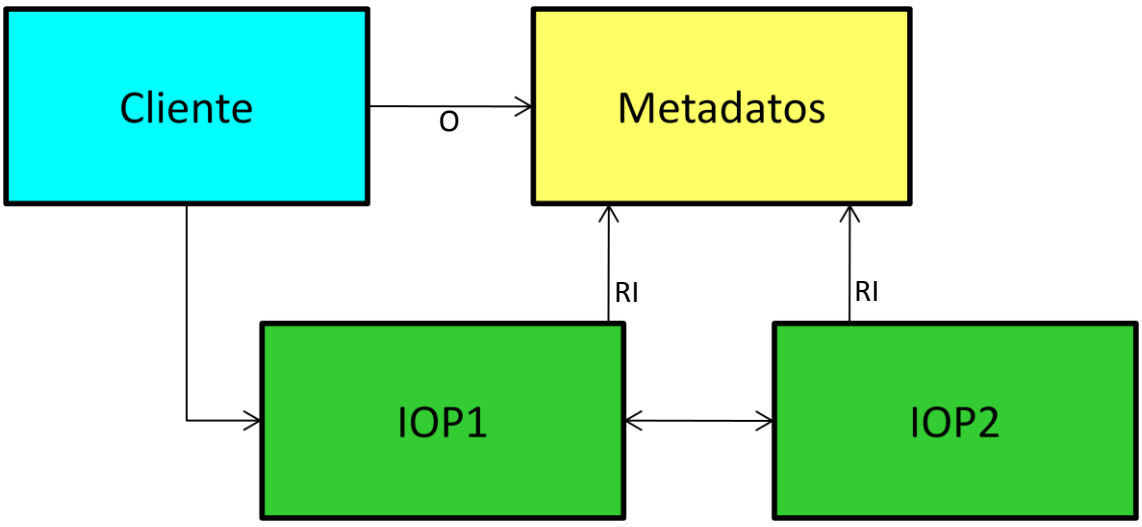


Figura 16. Llamadas RPC del sistema

- **Nodo Cliente**:
  - Solo se puede conectar a los nodos IOP de una red.
  - Se divide en **6 capas**: servicios FUSE, lógica del programa, cliente RPC nodo IOP, cliente RPC nodo de Metadatos, sistema de ficheros NFS y base de datos.

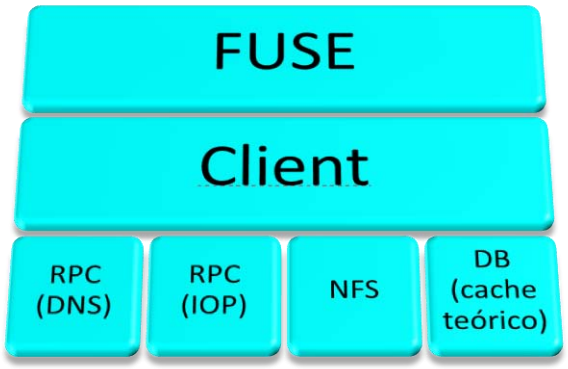


Figura 17. Capas nodo Cliente

  - Cuando una llamada RPC a un nodo IOP devuelva NULL (el servidor no responde) se considerara que el nodo está caído y se notificara al resto de nodos IOP de la red con la función RPC IOP caído.
  - Cuando una llamada RPC de escritura o lectura devuelve NULL (el servidor no responde) se realiza esta llamada directamente a disco.

- **Nodo IOP**:
  - Se considera que **pueden sufrir caídas o perdida de conexión**.
  - Se agrupan formando una red de IOP.
  - **Solo pueden formar parte de una red de IOP al mismo tiempo**.
  - Se identifican de forma univoca por el nombre de la red de IOP de la que forman parte y la posición que ocupan dentro de esta red.
  - Se divide en **6 capas**: servidor RPC, lógica del programa, base de datos, sistema de ficheros NFS, cliente RPC nodo IOP y cliente RPC nodo de Metadatos.

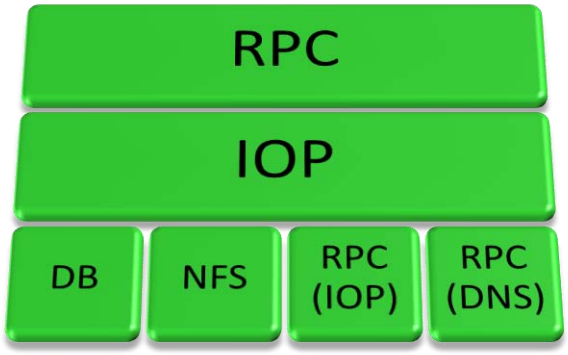


Figura 18. Capas nodo IOP

- Llamadas RPC del nodo IOP:
  - **IOP caído** (IC): notifica que un nodo IOP no responde.
  - **Abrir fichero** (A).
  - **Cerrar fichero** (C).
  - **Escribir bloque** (E).
  - **Leer bloque** (L).
  - **IOP activo** (IA): notifica que un nodo IOP se ha levantado.
  - **Guardar *backup*** (GB): copia de seguridad de un bloque de datos de otro nodo IOP.
  - **Eliminar *backup*** (EB): elimina una copia de seguridad de otro IOP

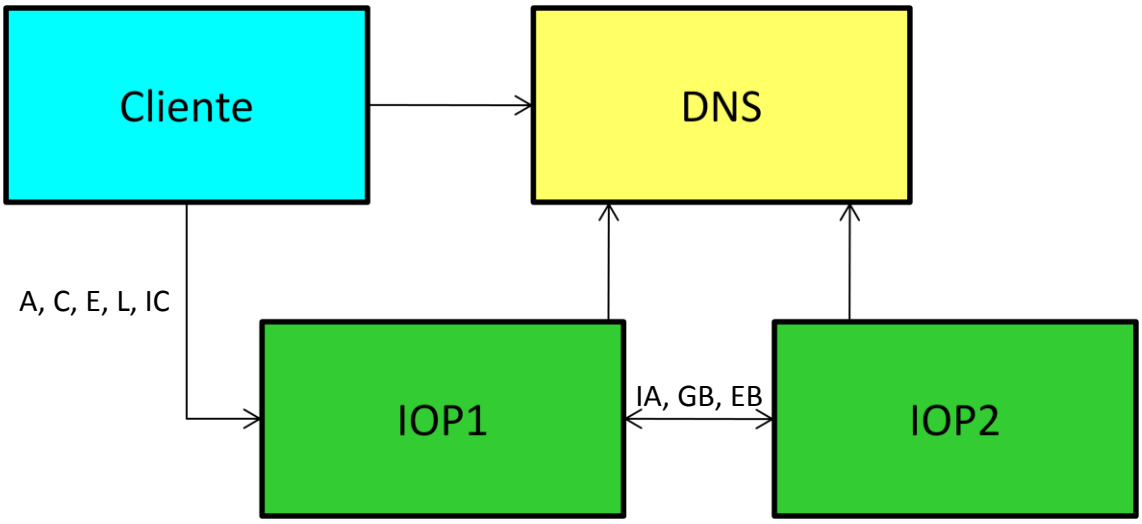


Figura 19. Llamadas RPC del sistema

- Al levantarse debe notificar al resto de nodos IOP de su red que está activo.
- Política de distribución: el 1º bloque de un fichero se almacenara siguiendo una función HASH del nombre del fichero.
- Los bloques de un fichero se almacenaran en los siguientes nodos IOP al nodo que contiene el 1º bloque.
- Los nodos IOP almacenan en su base de datos los bloques de las llamadas de escritura y de lectura.

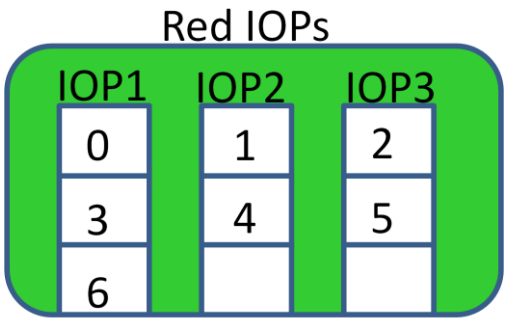
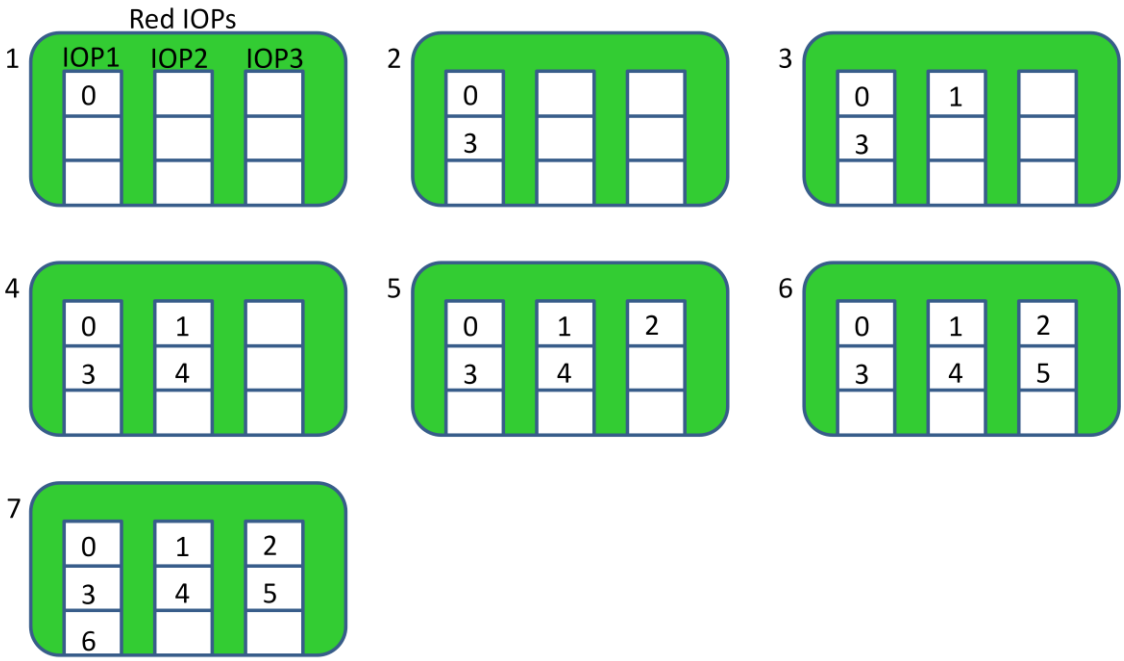


Figura 20. Fichero distribuido en bloques

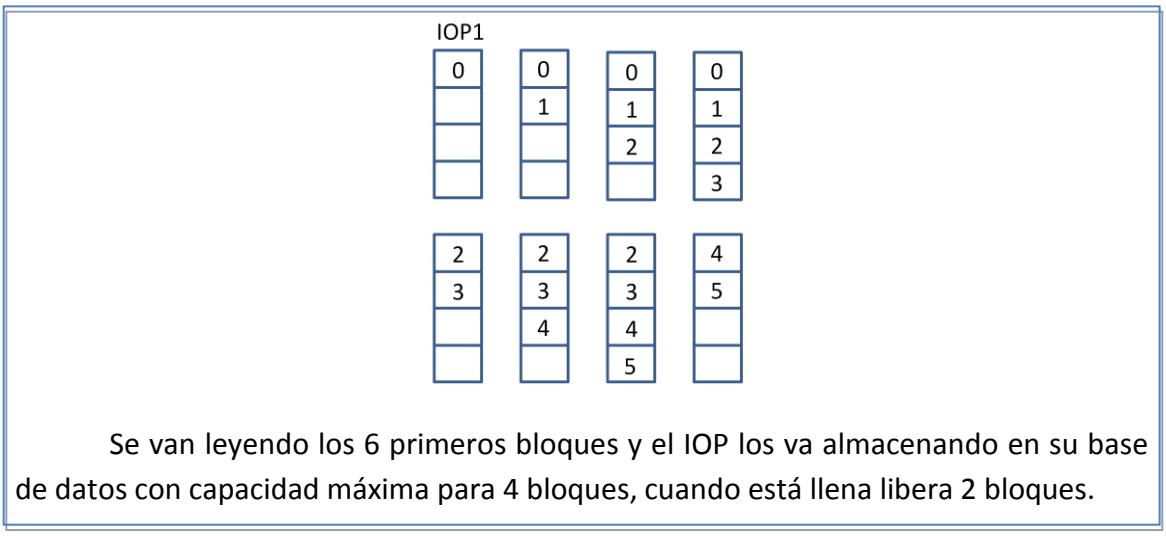
- La **base de datos tiene una capacidad finita**.
- La base de datos contiene **bloques de 3 tipos**:
  - **Sucios**: bloques que contienen datos que aun no han sido escritos en disco.
  - **Limpios**: bloques que contienen la misma información que el bloque en disco.
  - ***Backup***: bloque que pertenece a otro nodo IOP.
- Poseen **3 políticas de lectura**:
  - **Bajo demanda**: cuando el nodo recibe una llamada de lectura, carga en la base de datos el bloque solicitado y devuelve el bloque.
  - ***Background***: cuando el nodo recibe una llamada de lectura, carga en la base de datos el bloque solicitado y devuelve el bloque. En segundo plano carga los siguientes X bloques en la base de datos.
  - **Espera pasiva**: cuando el nodo recibe una llamada de lectura, carga en la base de datos los siguientes X bloques al solicitado y devuelve el bloque solicitado.



1. IOP1 recibe una petición de lectura del bloque 0 lo lee de disco y lo almacena en la base de datos.
2. El IOP1 en background almacena el siguiente bloque que le corresponde leer el bloque 3 lo lee de disco.
3. IOP2 recibe una petición de lectura del bloque 1 lo lee de disco y lo almacena en la base de datos.
4. El IOP2 en background almacena el siguiente bloque que le corresponde leer el bloque 4 lo lee de disco.
5. IOP3 recibe una petición de lectura del bloque 2 lo lee de disco y lo almacena en la base de datos.
6. El IOP3 en background almacena el siguiente bloque que le corresponde leer el bloque 5 lo lee de disco.
7. IOP1 recibe una petición de lectura del bloque 3, al estar almacenado en la base de datos no tiene que leerlo de disco.
8. El IOP1 en background almacena el siguiente bloque que le corresponde leer el bloque 6 lo lee de disco.

Figura 21. Lectura de 4 bloques, 3 IOP, lectura background = 1

- Poseen **3 políticas de escritura**:
  - **Escritura demorada**: el IOP cada X segundos escribe los bloques sucios a disco, estos bloques pasan a ser bloques limpios.
  - **Escritura a disco**: cuando un nodo IOP recibe una llamada de escritura escribe el bloque directamente a disco. La base de datos nunca contiene bloques de tipo sucio.
  - **Escritura X IOP**: cuando un nodo IOP recibe una llamada de escritura almacena el bloque en la base de datos de los X nodos IOP vecinos. Los nodos vecinos almacenan el bloque de tipo *backup*.
- Poseen **2 políticas de reemplazamiento**:
  - **Primero bloques sucios**: cuando la base de datos está llena se eliminan X bloques sucios, escribiendo estos a disco y notificando a los nodos IOP vecinos que ya no tienen que almacenar la copia *backup*. Si no hay suficientes bloques sucios que eliminar se eliminan los limpios.
  - **Primero bloques limpios**: cuando la base de datos está llena se eliminan X bloques limpios. Si no hay suficientes bloques limpios que eliminar se eliminan los sucios, escribiendo estos a disco y notificando a los nodos IOP vecinos que ya no tienen que almacenar la copia *backup*.



Se van leyendo los 6 primeros bloques y el IOP los va almacenando en su base de datos con capacidad máxima para 4 bloques, cuando está llena libera 2 bloques.

Figura 22. **Leer 6 bloques, 1 IOP, política reemplazamiento = 2**

- Cuando recibe una llamada de que otro nodo IOP esta caído escribe en disco todos los bloques de tipo *backup* que tenga almacenados del nodo caído directamente a disco.

- **Llamadas POSIX** (suposición 2 nodos IOP en la red):
    - **Abrir**: el nodo Cliente llama a la función RPC abrir fichero de cada nodo IOP de la red, si algún nodo de devuelve que no se puede abrir ya no se realiza ninguna llamada mas.
    - **Cerrar**: el nodo Cliente llama a la función RPC cerrar fichero de cada nodo IOP de la red.
    - **Leer**: el nodo Cliente llama a la función RPC leer bloque de cada nodo IOP involucrado en la lectura, usa un proceso ligero por cada nodo IOP.
    - **Escribir**: el nodo Cliente llama a la función RPC escribir bloque de cada nodo IOP involucrado en la lectura, usa un proceso ligero por cada nodo IOP.

4.2 Diagrama de clase

En este apartado se mostraran los diagramas de clase de los 3 nodos que forman el sistema, así como una breve descripción de cada clase.

4.2.1 Nodo Cliente

Este es el diagrama de clases del nodo Cliente, se ha omitido todo el código autogenerado mediante la herramienta rpcgen para simplificar el diagrama. Las diferentes clases se describirán con mayor detalle en las secciones siguientes.

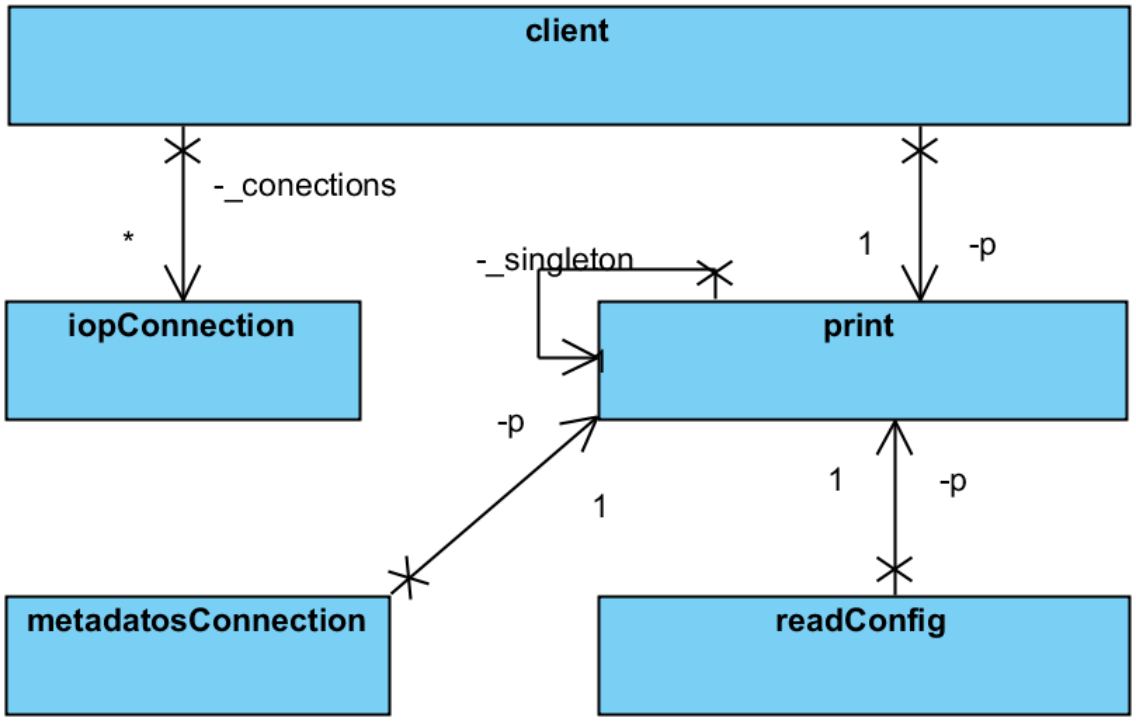


Diagrama de clase 1. Nodo Cliente

4.2.1.1 Clase client

Esta clase contiene la lógica del nodo Cliente, abre los ficheros en los nodos IOP, los cierra, lee los bloques de un fichero en paralelo en los diferentes nodos IOP, escribe los bloques de un fichero en paralelo en los diferentes nodos IOP.

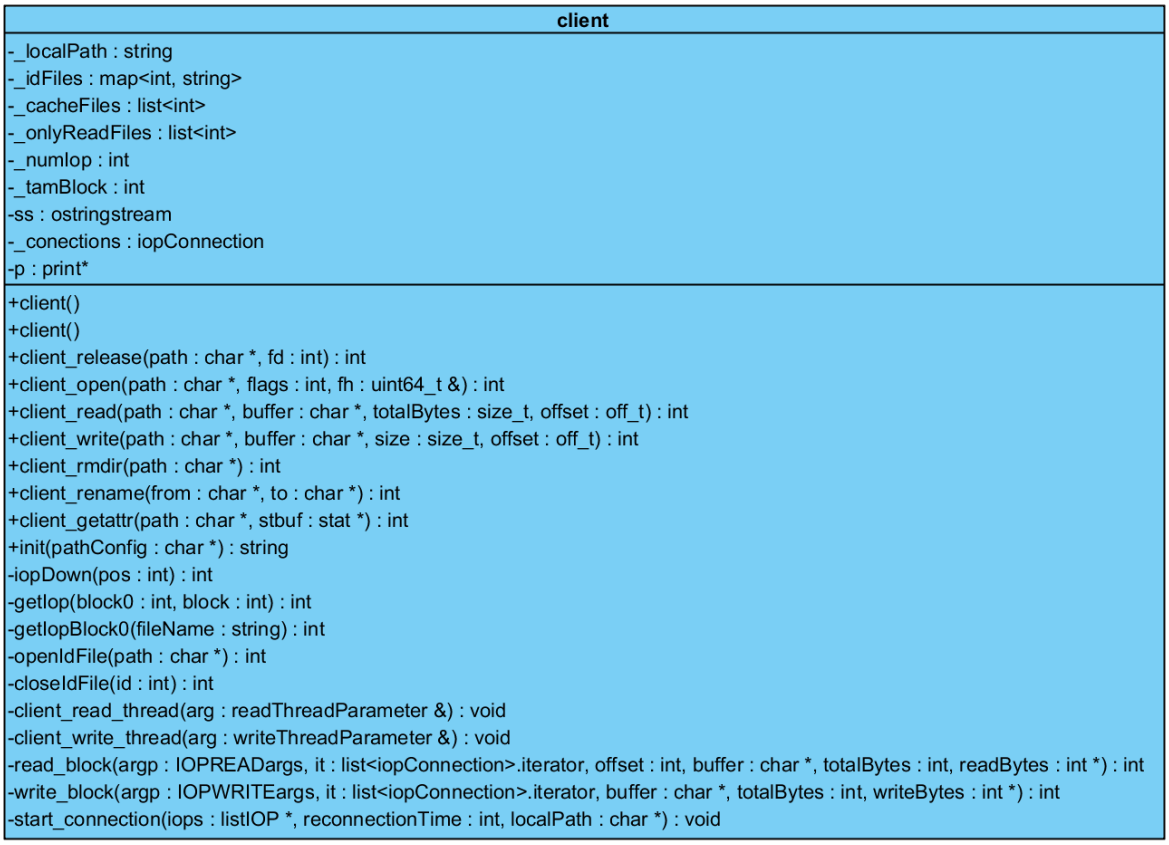


Diagrama de clase 2. Clase client

4.2.1.2 Clase metadatosConnection

Esta clase realiza las llamadas remotas RPC al servidor del nodo de Metadatos, también gestiona la reconexión con el mismo en caso de pérdida de conexión.

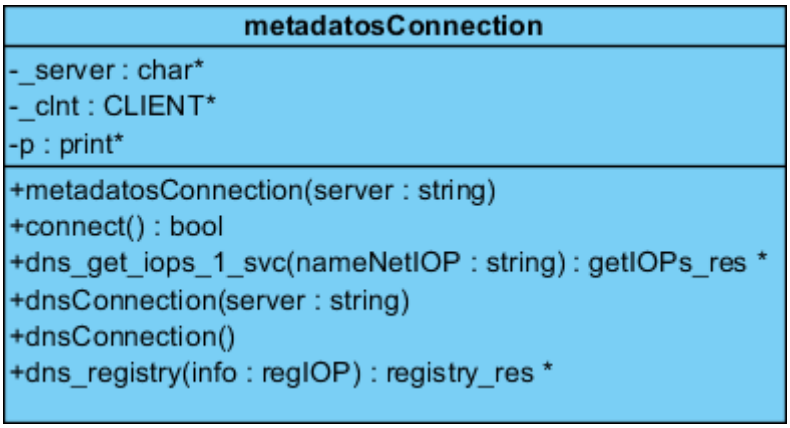


Diagrama de clase 3. Clase dnsConnection

4.2.1.3 Clase iopConnection

Esta clase realiza las llamadas remotas RPC al servidor del nodo IOP, también gestiona la reconexión con el mismo en caso de pérdida de conexión.

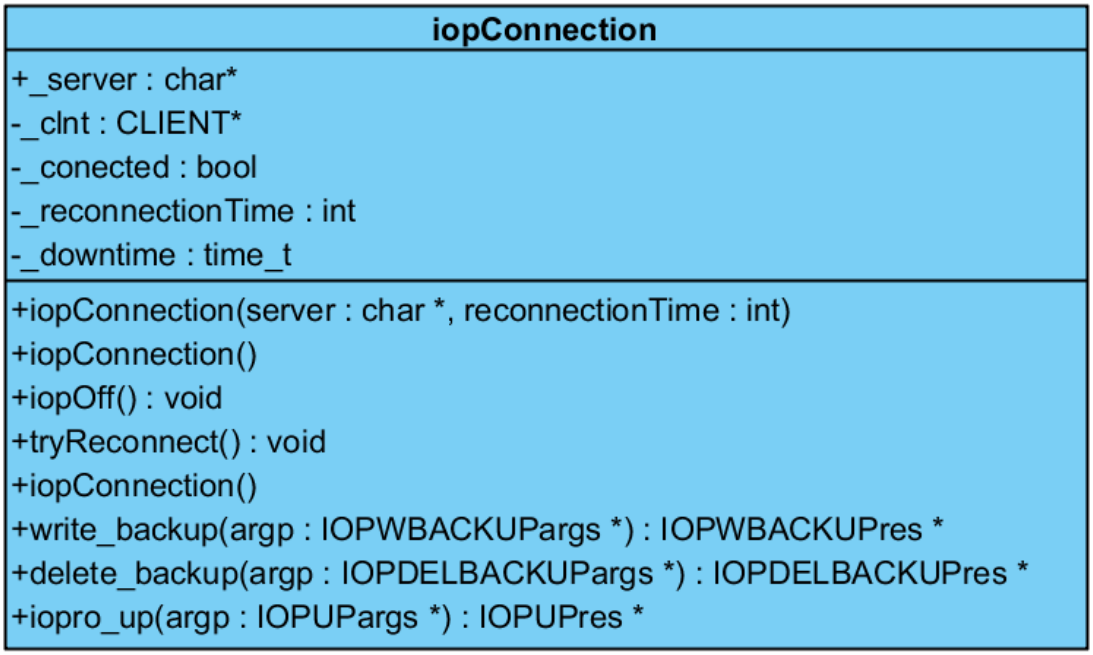


Diagrama de clase 4. Clase iopConnection

4.2.1.4 *Clase print*

Se trata de una clase auxiliar que permite una impresión jerárquica por pantalla en función del nivel de anidamiento de la llamada. Si un proceso ligero realiza una llamada a *print* mientras se estaba imprimiendo la traza de otra función, este mensaje se imprimirá al final. La figura 22 ilustra un ejemplo.

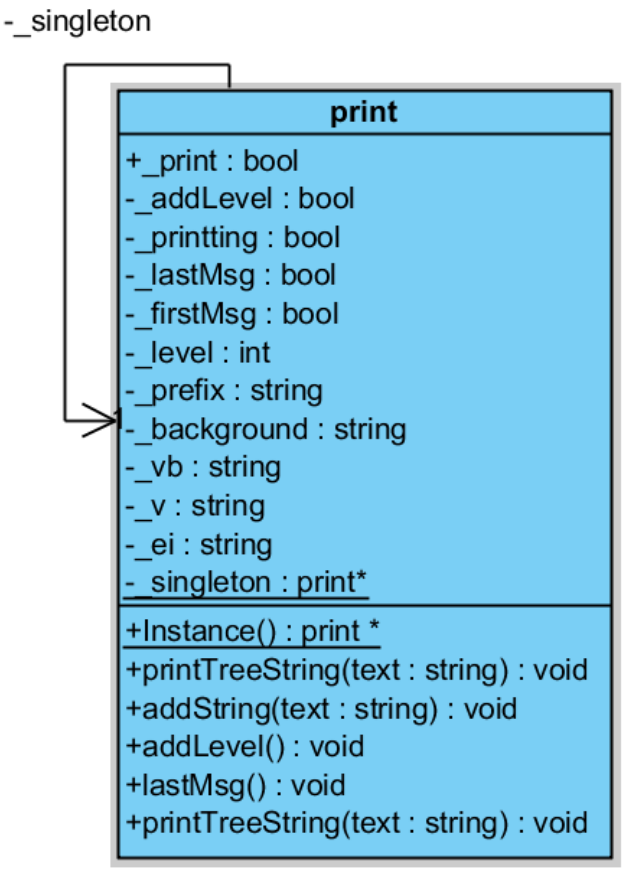


Diagrama de clase 5. Clase print

```
start_server()
├─start_service()
│  ├─DB started in memory.
│  └─finish start_service()
└─finish start_server()
```

Figura 23. Ejemplo print

Consultar Apéndice V. Manual print para más información.

4.2.1.5 *Clase readConfig*

Clase encargada de leer el fichero de configuración del nodo Cliente y almacenar los valores de dicho fichero en variables. Si alguna variable imprescindible del fichero de configuración no se encuentra en el fichero imprimirá un mensaje de alerta indicando la variable que no se ha encontrado, también gestiona que el fichero de configuración contenga variables con el valor correcto.

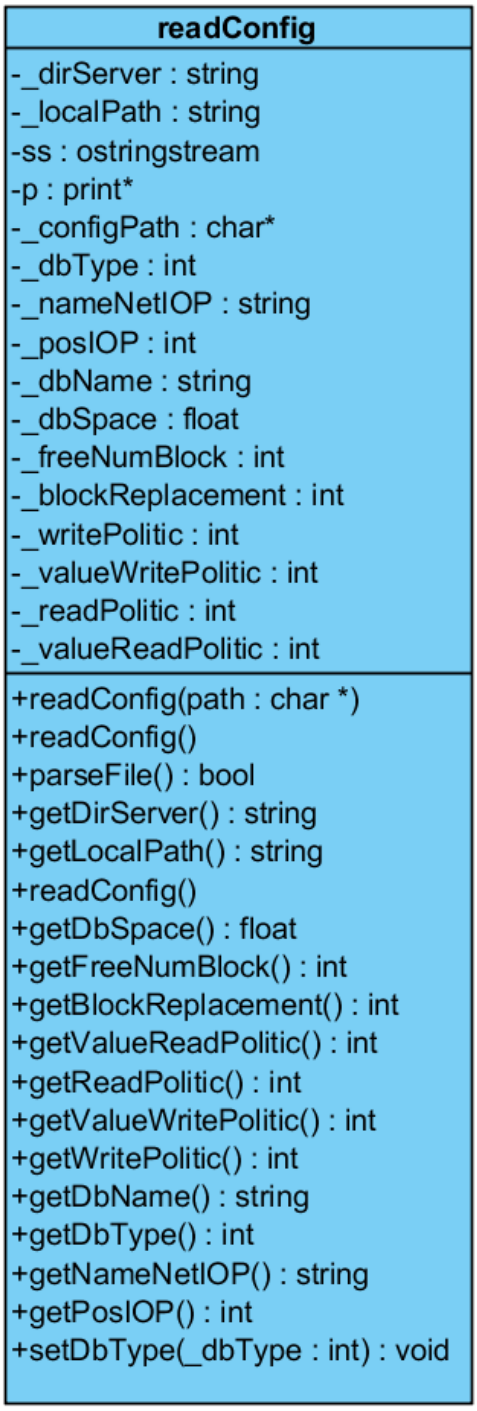


Diagrama de clase 6. Clase readConfig

4.2.2 Nodo IOP

Este es el diagrama de clases del nodo IOP, se ha omitido todo el código autogenerado mediante la herramienta rpcgen para simplificar el diagrama.

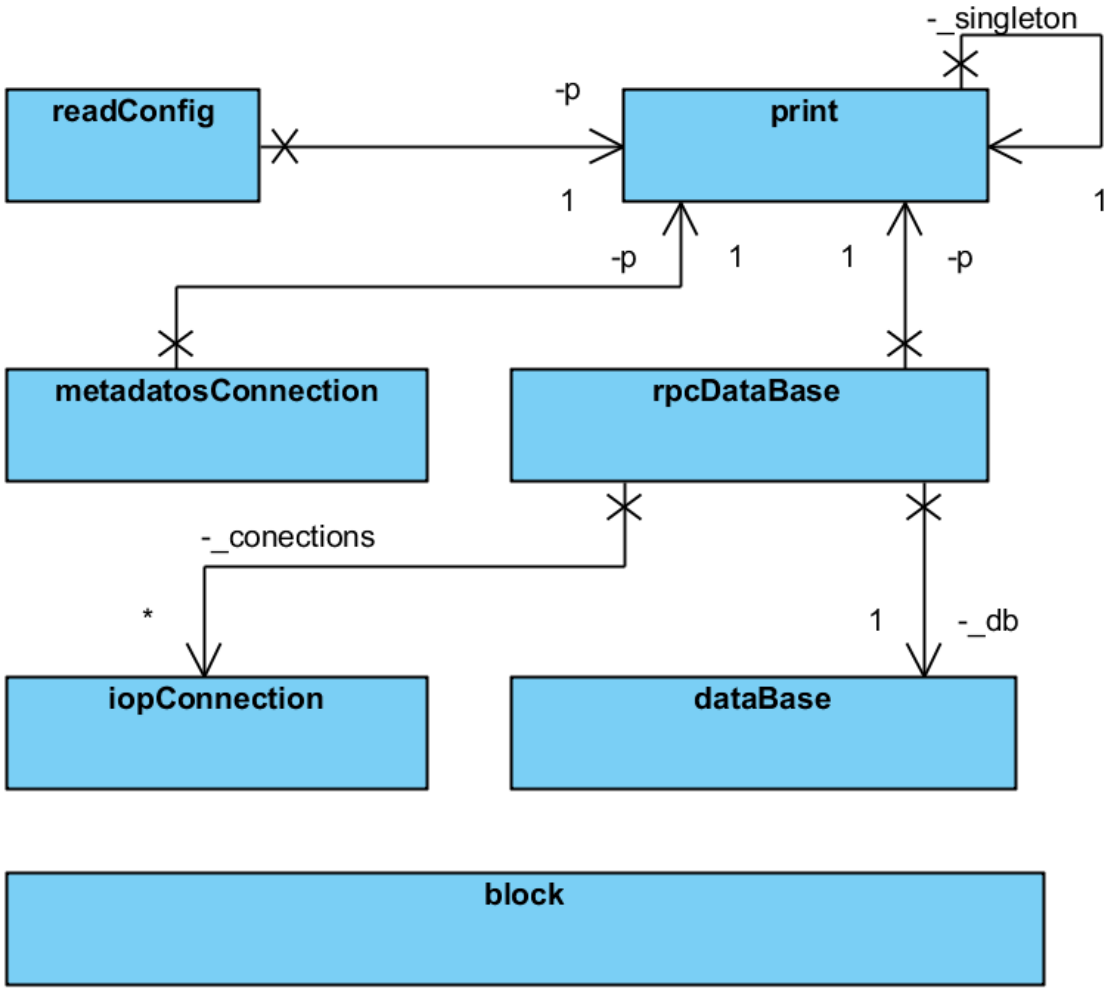


Diagrama de clase 7. Nodo IOP

4.2.2.1 Clase block

Crea las entradas para la base de datos del nodo IOP. A partir de un nombre de fichero y un offset crea la clave para la base de datos. El valor asociado a esta clave es el bloque de datos del fichero

También transforma una entrada de la base de datos a un nombre de fichero un offset y un bloque de datos.

La clave de la base de datos solo puede ser una secuencia char* y los datos también deben ser una secuencia char*.

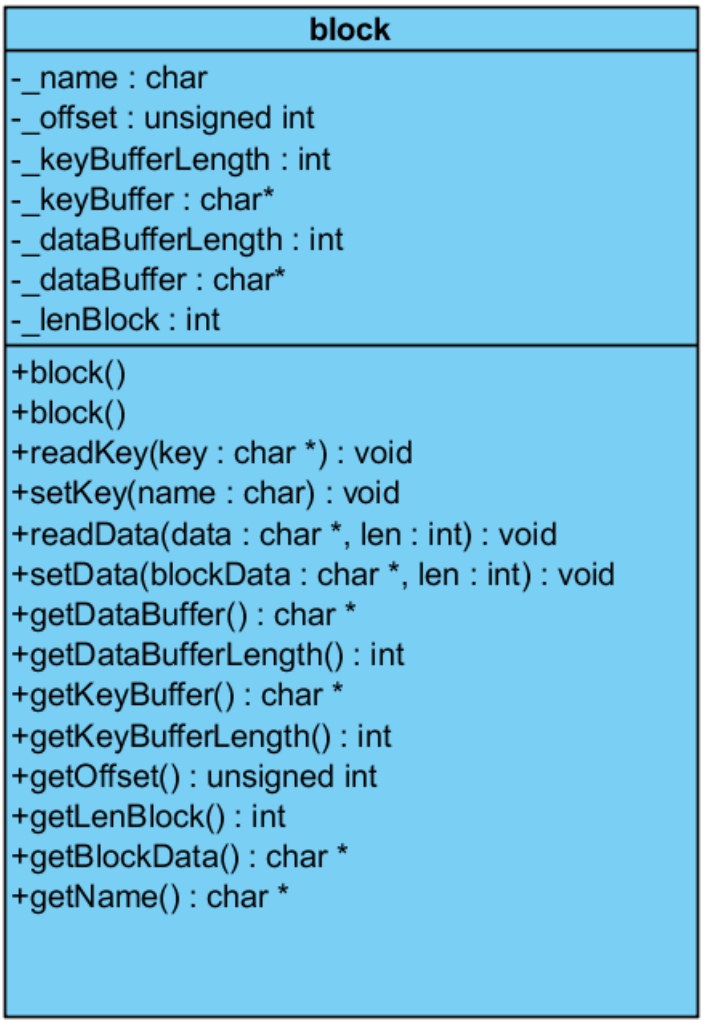


Diagrama de clase 8. Clase block

4.2.2.2 Clase dataBase

Gestiona la base de datos, permite cargar, leer o eliminar una entrada de la base de datos.

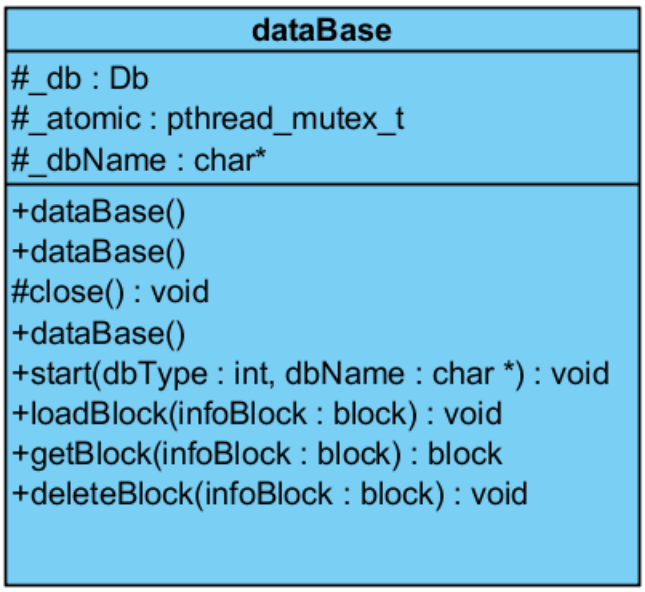


Diagrama de clase 9. Clase dataBase

4.2.2.3 Clase rpcDataBase

Esta clase contiene la lógica del nodo IOP, responde a las peticiones de los nodos Cliente, leyendo bloques de disco y de la base de datos, ejecuta las diferentes políticas de lectura y escritura y gestiona los backup de bloques de otros nodos IOP.

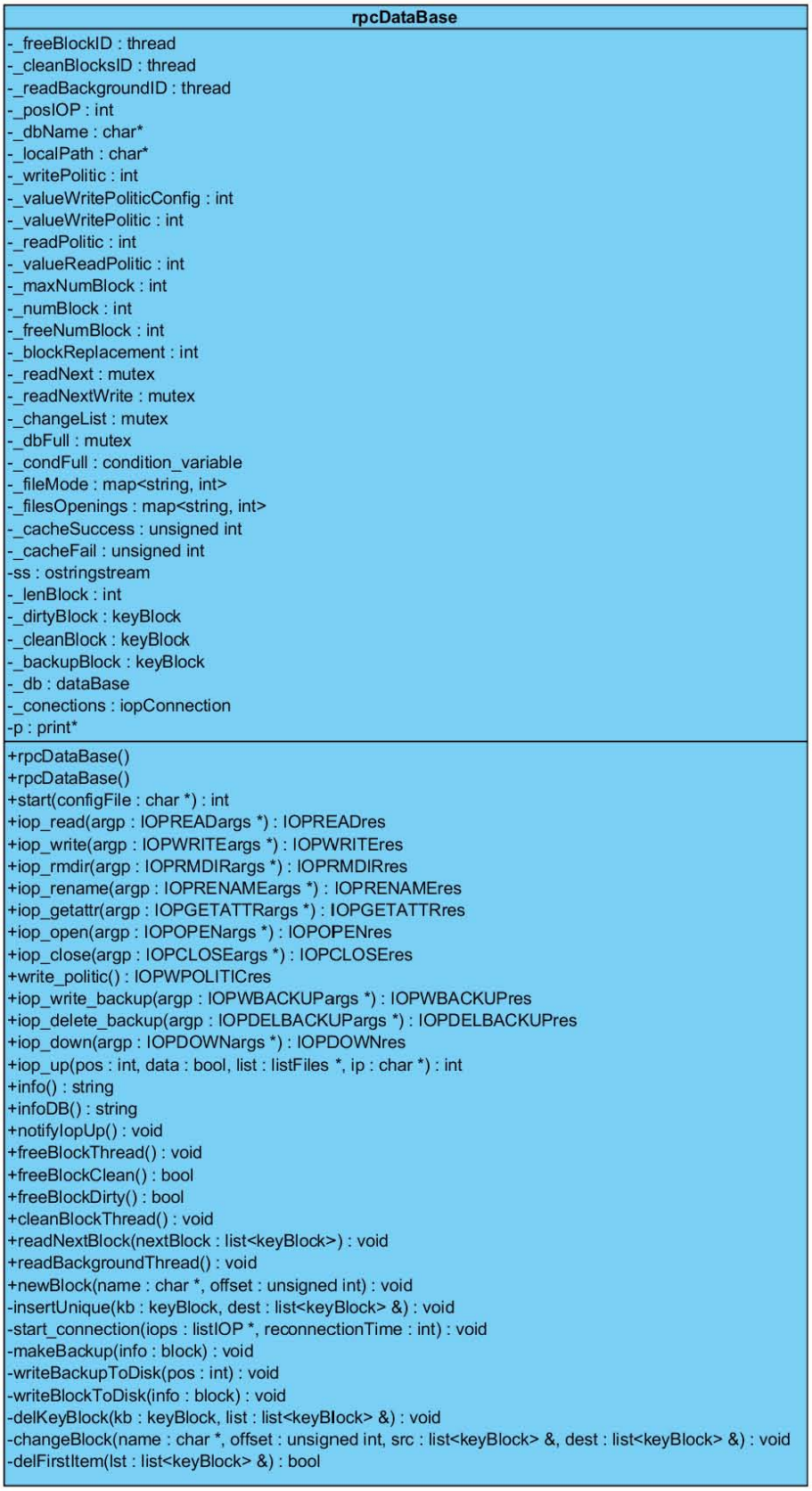


Diagrama de clase 10. Clase rpcDataBase

4.2.2.4 Clase readConfig

Clase encargada de leer el fichero de configuración del nodo IOP y almacenar los valores de dicho fichero en variables. Si alguna variable imprescindible del fichero de configuración no se encuentra en el fichero imprimirá un mensaje de alerta indicando la variable que no se ha encontrado, también gestiona que el fichero de configuración contenga variables con el valor correcto.

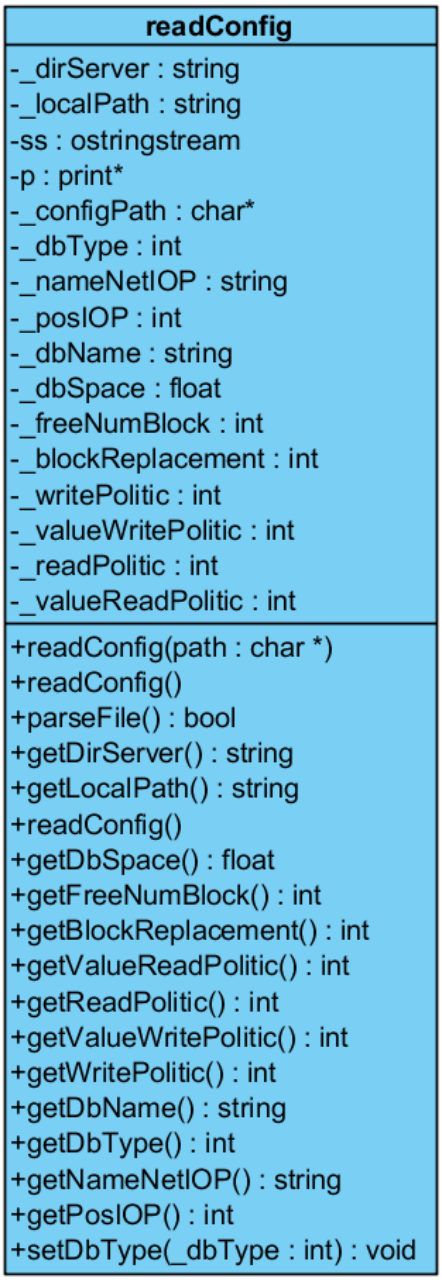


Diagrama de clase 11. Clase readConfig

4.2.2.5 Clase metadatosConnection

Esta clase realiza las llamadas remotas RPC al servidor del nodo de Metadatos, también gestiona la reconexión con el mismo en caso de pérdida de conexión.

Ver sección 4.2.1.2.

4.2.2.6 Clase iopConnection

Esta clase realiza las llamadas remotas RPC al servidor del nodo IOP, también gestiona la reconexión con el mismo en caso de pérdida de conexión.

Ver sección 4.2.1.3.

4.2.2.7 Clase print

Clase auxiliar que permite una impresión jerárquica por pantalla en función del nivel de anidamiento de la llamada. Si un proceso ligero realiza una llamada a *print* mientras se estaba imprimiendo la traza de otra función, este mensaje se imprimirá al final.

Ver sección 4.2.1.4.

4.2.3 Nodo de Metadatos

Este es el diagrama de clases del nodo de Metadatos, se ha omitido todo el código autogenerado mediante la herramienta rpcgen para simplificar el diagrama.

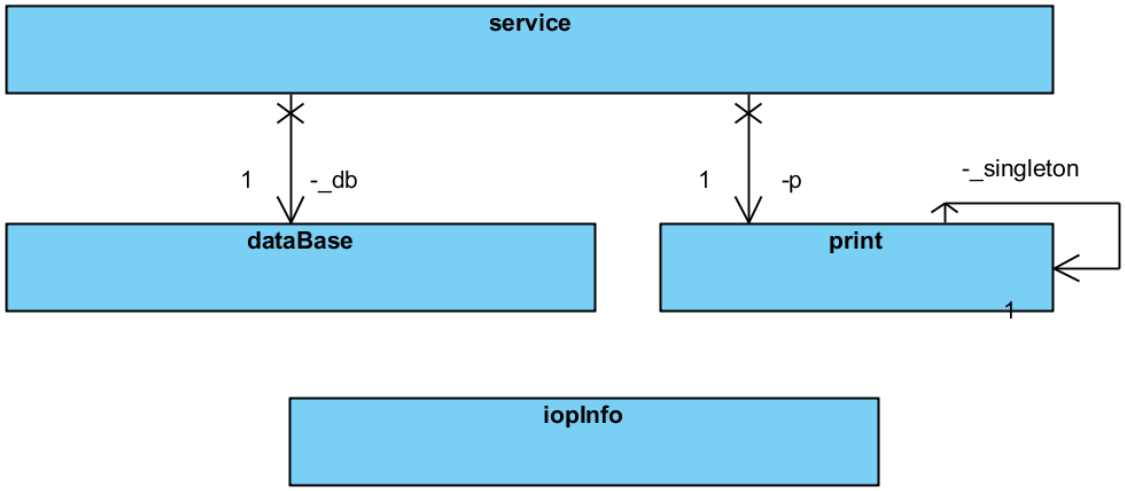


Diagrama de clase 12. Nodo de Metadatos

4.2.3.1 *Clase Service*

Esta clase contiene la lógica del nodo de Metadatos, responde a las peticiones de los nodos Cliente, leyendo de la base de datos todos los nodos IOP pertenecientes a una red y devolviendo dicha información y también responde a las peticiones de los nodos IOP, registrando un nuevo nodo en una red y devolviendo la información de los nodos anteriormente registrados en dicha red.

service
-_db : dataBase -p : print*
+service() +service() +registry_iop(nameNetIOP : char *, posIOP : char, ip : char *, list : listIOP *) : int +get_iops(list : listIOP *, nameNetIOP : char *) : int +start_service() : void

Diagrama de clase 13. Clase service

4.2.3.2 Clase iopInfo

Crea las entradas para la base de datos del nodo DNS. A partir de un nombre de red de IOP y una posición de un nodo IOP crea la clave para la base de datos. El valor asociado a esta clave es la dirección IP del nodo IOP

También transforma una entrada de la base de datos a un nombre de red, una posición y una dirección IP.

La clave de la base de datos solo puede ser una secuencia char* y los datos también deben ser una secuencia char*.

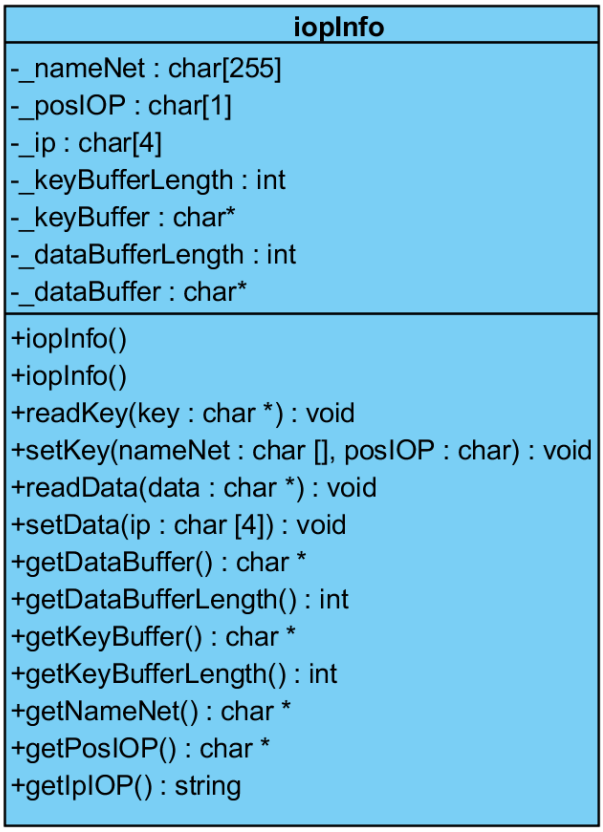


Diagrama de clase 14. Clase iopInfo

4.2.3.3 Clase print

Clase auxiliar que permite una impresión jerárquica por pantalla en función del nivel de anidamiento de la llamada. Si un proceso ligero realiza una llamada a *print* mientras se estaba imprimiendo la traza de otra función, este mensaje se imprimirá al final.

Ver sección 4.2.1.4.

4.2.3.4 Clase dataBase

Gestiona la base de datos, permite cargar, leer o eliminar una entrada de la base de datos.

Ver sección 4.2.2.2.

4.3 Diagrama entidad relación

En este apartado de mostraran los diagramas entidad relación de las dos bases de datos que forman parte del proyecto. La base de datos del nodo IOP y la base de datos del nodo de Metadatos.

4.3.1 Base de datos nodo IOP

Nombre del fichero al que pertenece el bloque, offset del bloque respecto al comienzo del fichero y valor del bloque de datos.

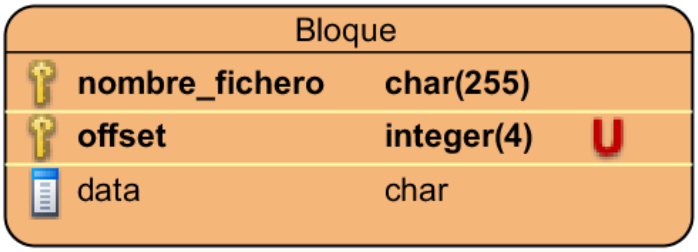


Diagrama entidad relación 1. Base de datos nodo IOP

4.3.2 Base de datos nodo de Metadatos

Nombre de la red de nodos IOP, posición que ocupa el nodo IOP dentro de la red, y dirección IP del nodo IOP.

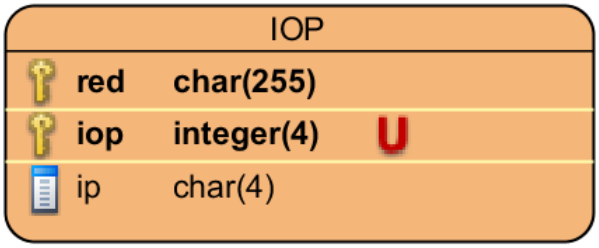


Diagrama entidad relación 2. Base de datos nodo de Metadatos

4.4 Diagramas de actividad

En esta sección se mostraran los diagramas de actividad de las funciones más relevantes del proyecto.

4.4.1 Nodo Cliente

El siguiente diagrama de actividad muestra como se cierra un fichero por parte de un nodo cliente.

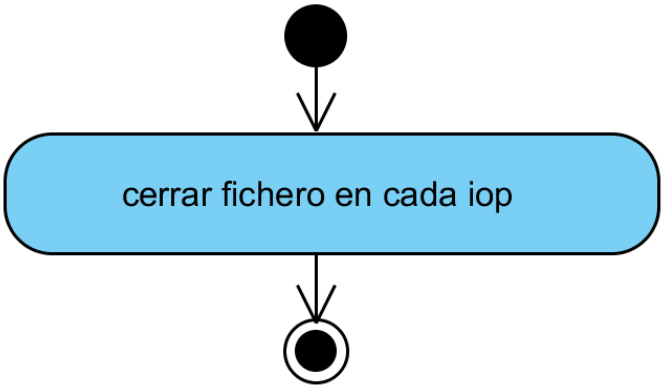


Diagrama de actividad 1. Nodo Cliente Cerrar fichero

El siguiente diagrama de actividad muestra como se abre un fichero por parte de un nodo cliente.

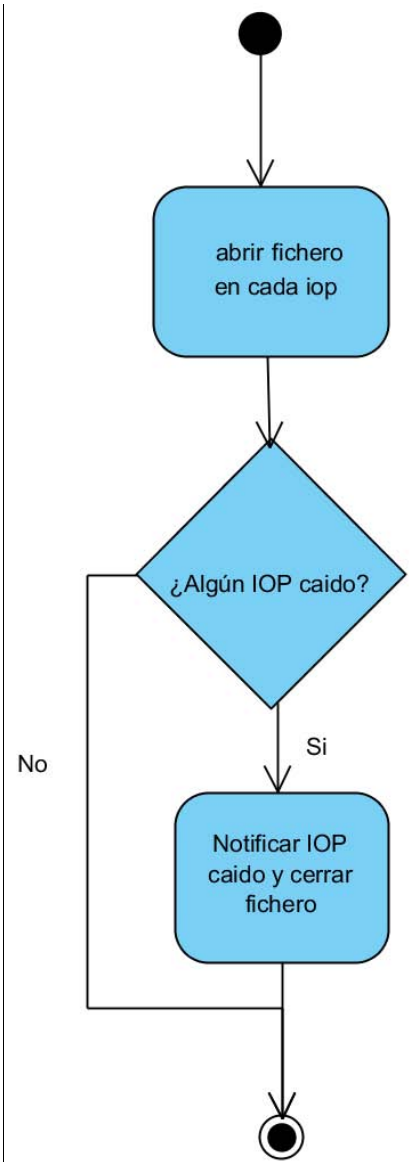


Diagrama de actividad 2. Nodo Cliente abrir fichero

El siguiente diagrama de actividad muestra como un nodo cliente lee un fichero.

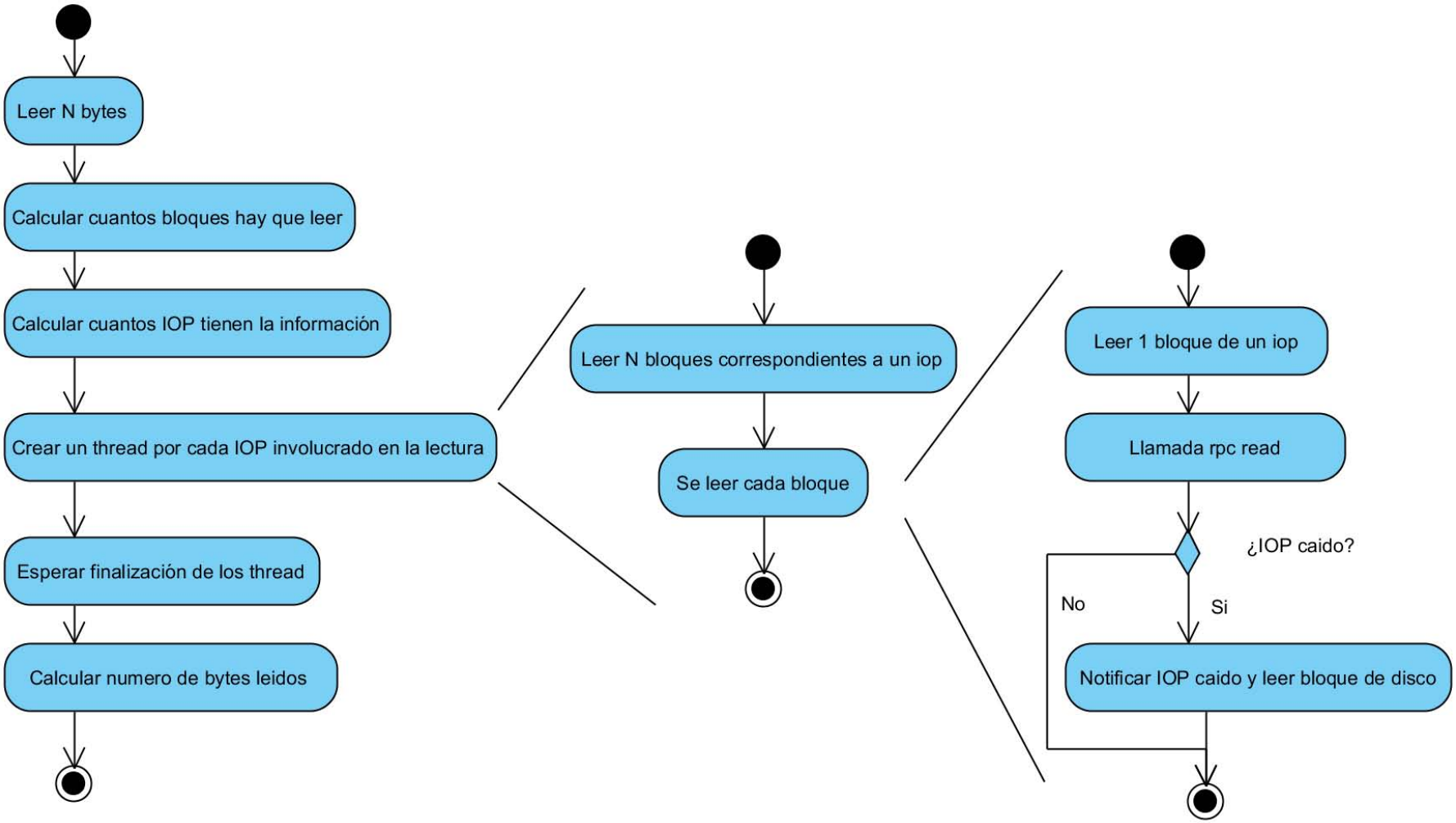


Diagrama de actividad 3. Nodo Cliente leer fichero

El siguiente diagrama de actividad muestra como se escribe un fichero por parte de un nodo cliente.

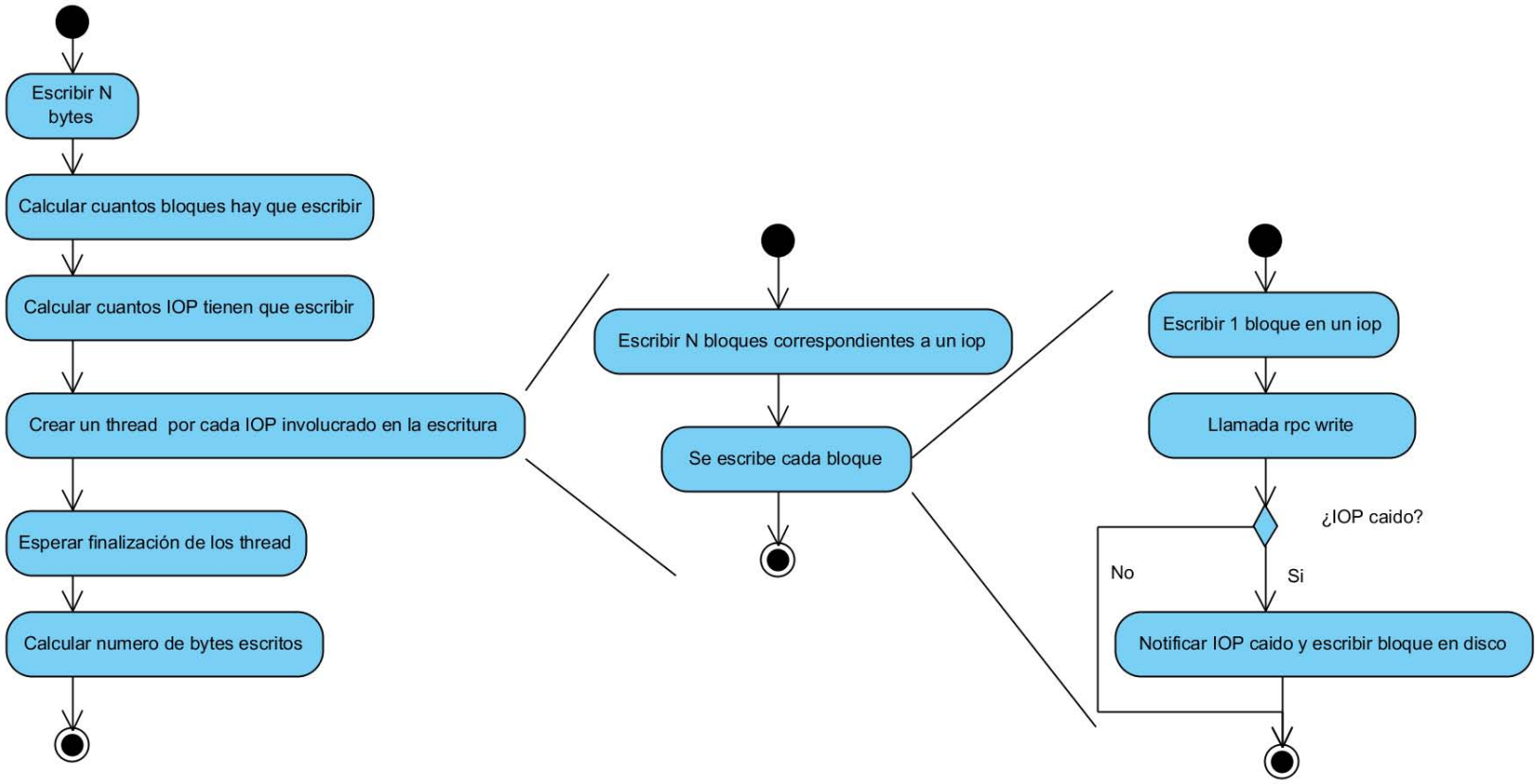


Diagrama de actividad 4. Nodo Cliente escribir fichero

4.4.2 Nodo IOP

El siguiente diagrama de actividad muestra como se abre un fichero por parte de un nodo IOP.

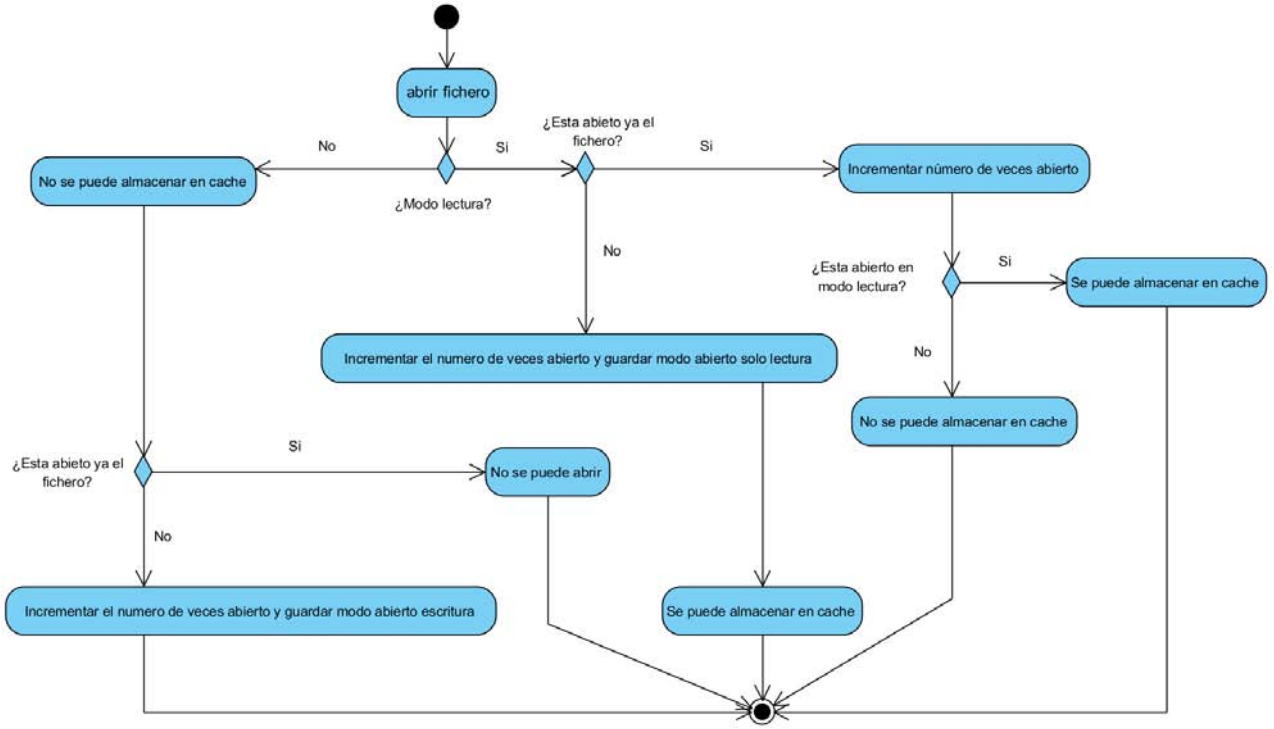


Diagrama de actividad 5. Nodo IOP abrir fichero

El siguiente diagrama de actividad muestra como se cierra un fichero por parte de un nodo IOP.

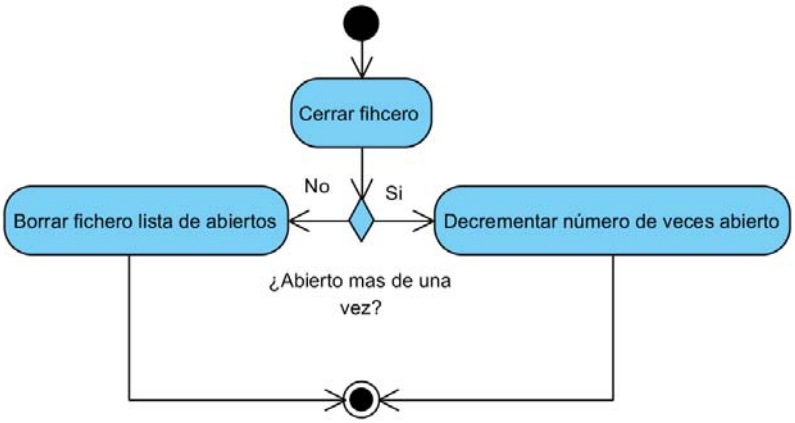


Diagrama de actividad 6. Nodo IOP cerrar fichero

El siguiente diagrama de actividad muestra como se lee un bloque en un nodo IOP.

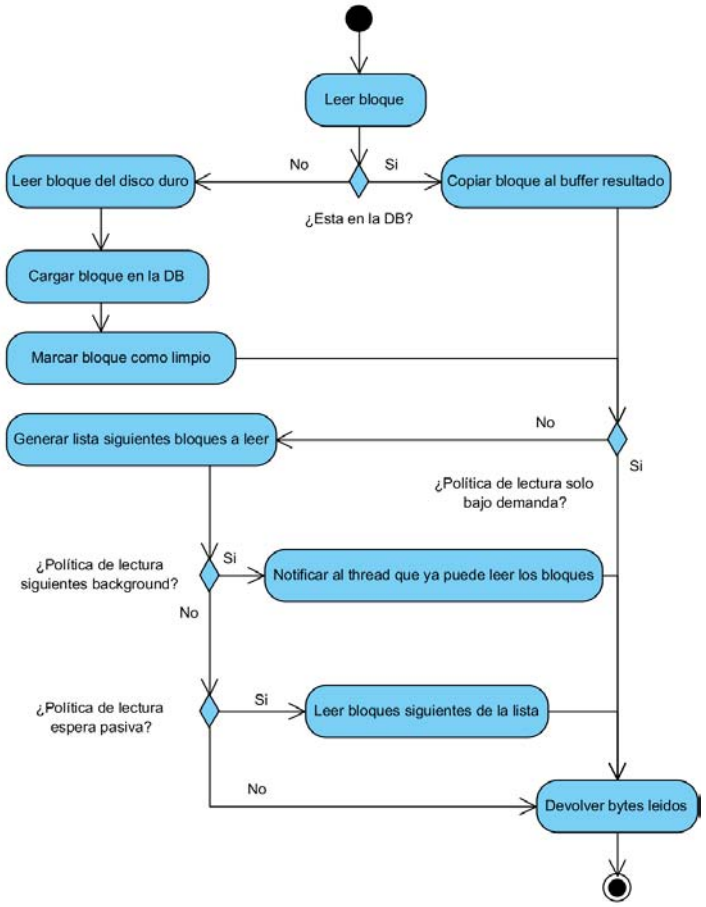


Diagrama de actividad 7. Nodo IOP leer bloque

El siguiente diagrama de actividad muestra como se escribe un bloque en un nodo IOP.

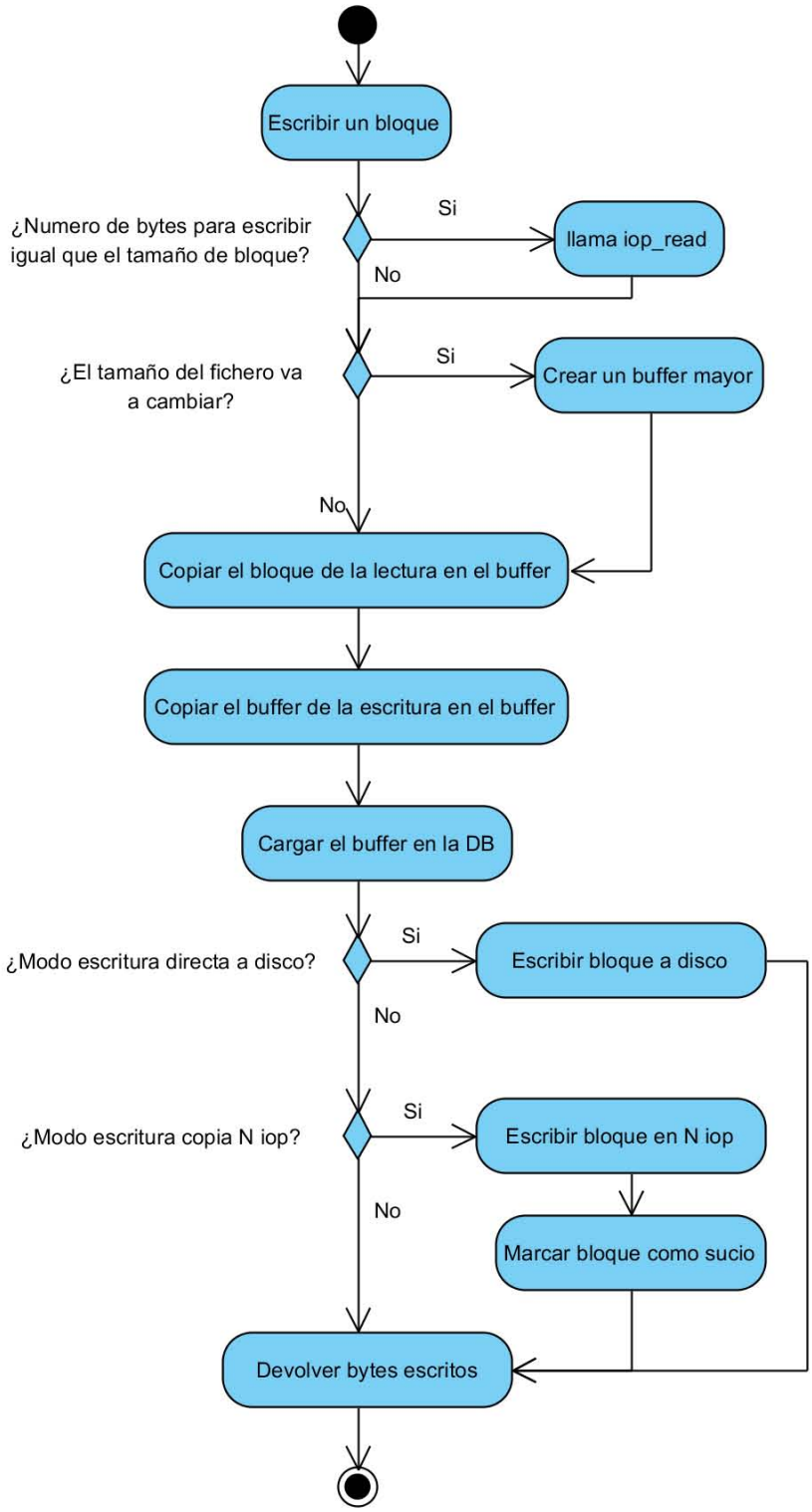


Diagrama de actividad 8. Nodo IOP escribir bloque

4.5 Diagramas de secuencia

En esta sección se mostraran los diagramas de actividad de las funciones más relevantes del proyecto. Se muestran solo funciones de alto nivel, sintetizando en algunos casos varias llamadas en una sola para simplificar el diagrama.

4.5.1 Nodo Cliente

En el siguiente diagrama se muestra como se inicia el nodo Cliente.

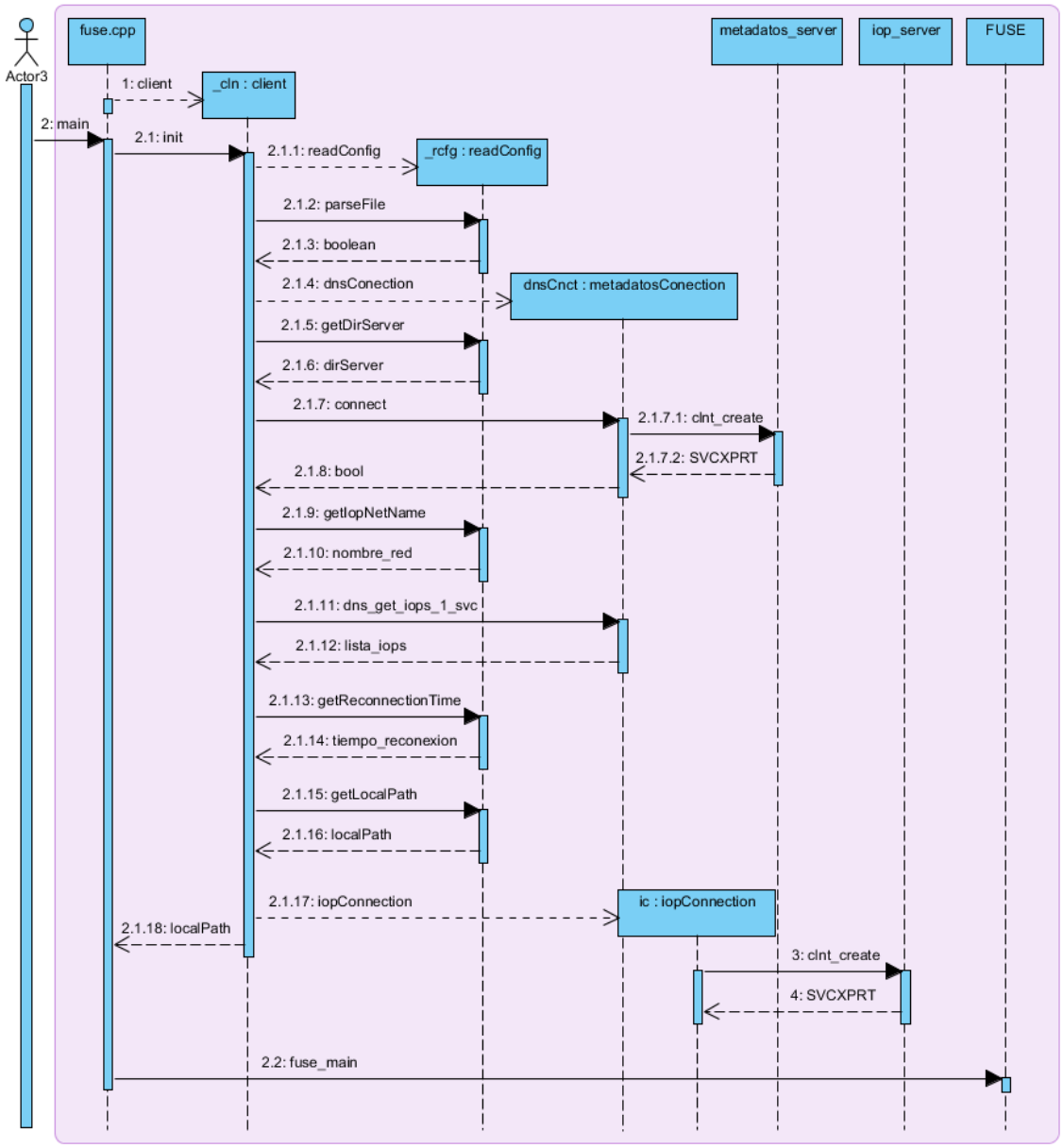


Diagrama de secuencia 1. Inicio Nodo Cliente

En el siguiente diagrama se muestra como un nodo Cliente abre un fichero en una red con 2 nodos IOP. Para ello llama a la función abrir fichero de cada nodo IOP.

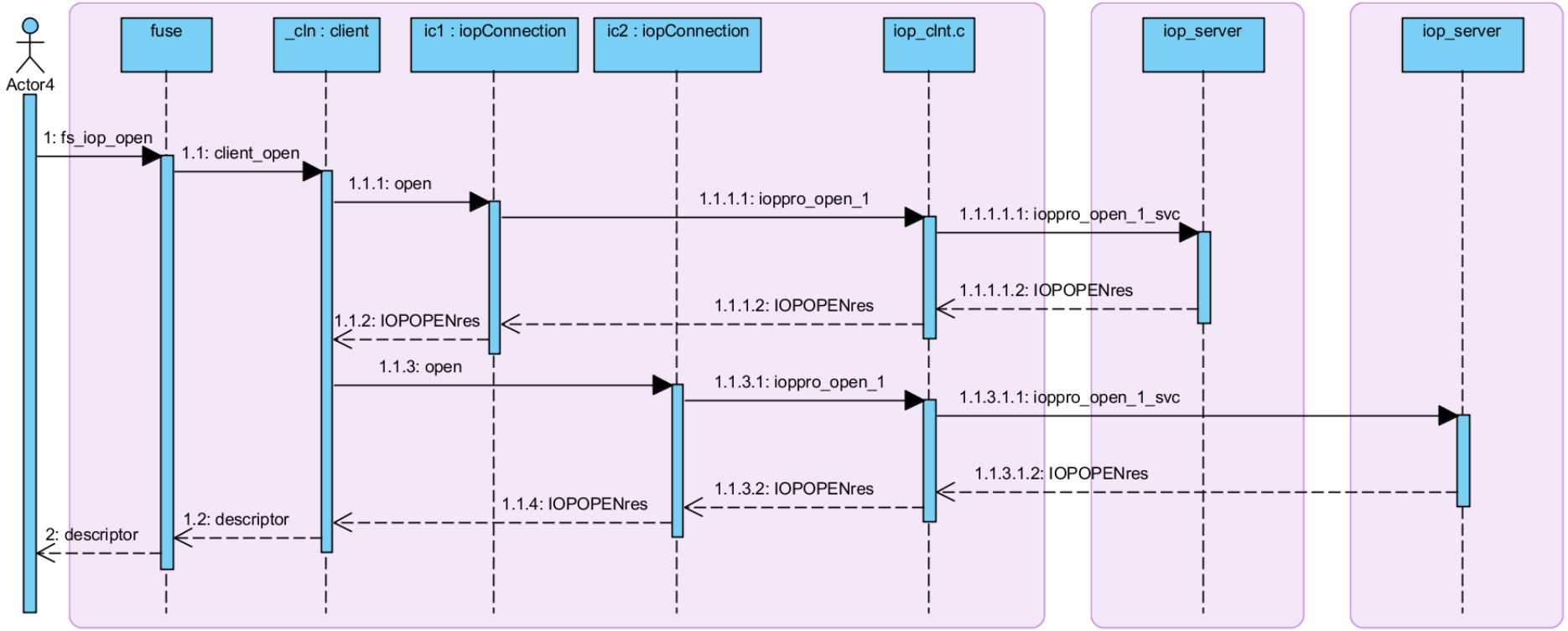


Diagrama de secuencia 2. Nodo Cliente abrir fichero

En el siguiente diagrama se muestra como un nodo Cliente abre un fichero en una red con 2 nodos IOP. Para ello llama a la función abrir fichero de cada nodo IOP. El segundo nodo IOP no responde por lo que se considera caído y se notifica al primer nodo IOP que hay un nodo caído.

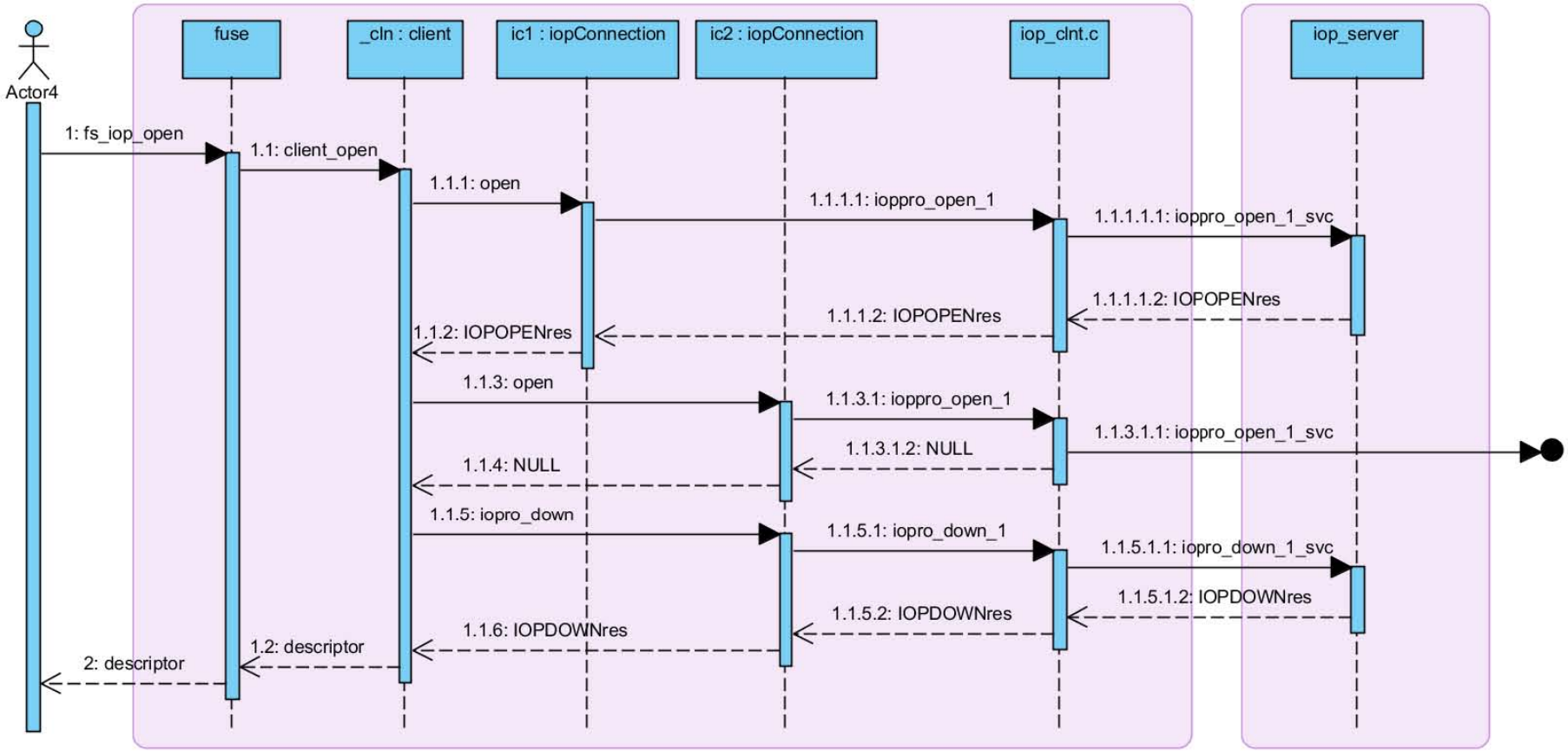


Diagrama de secuencia 3. Nodo Cliente abrir fichero iop caido

En el siguiente diagrama se muestra como un nodo Cliente cierra un fichero en una red con 2 nodos IOP. Se llama a la función cerrar fichero de cada nodo IOP.

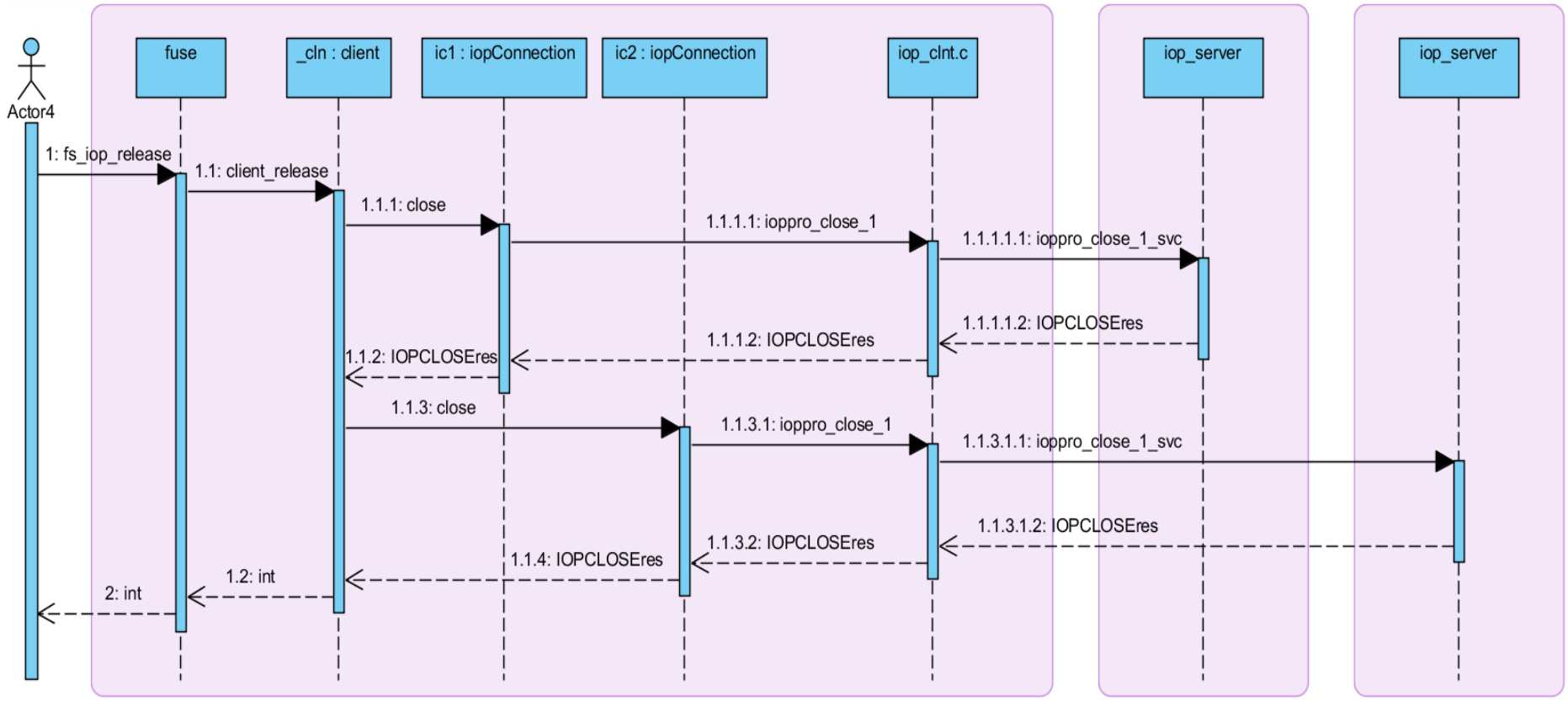


Diagrama de secuencia 4. Nodo Cliente cerrar fichero

En el siguiente diagrama, se muestra como un nodo Cliente lee dos bloques de un fichero en una red con 2 nodos IOP. Crea 2 proceso ligero que se encargan de leer los bloques necesarios en cada nodo de la red de IOP. Como el segundo thread no puede contactar con el nodo IOP se considera caído y se notifica a los otros nodos activos de la red. El bloque que debía ser leído del nodo IOP caído se lee directamente de disco.

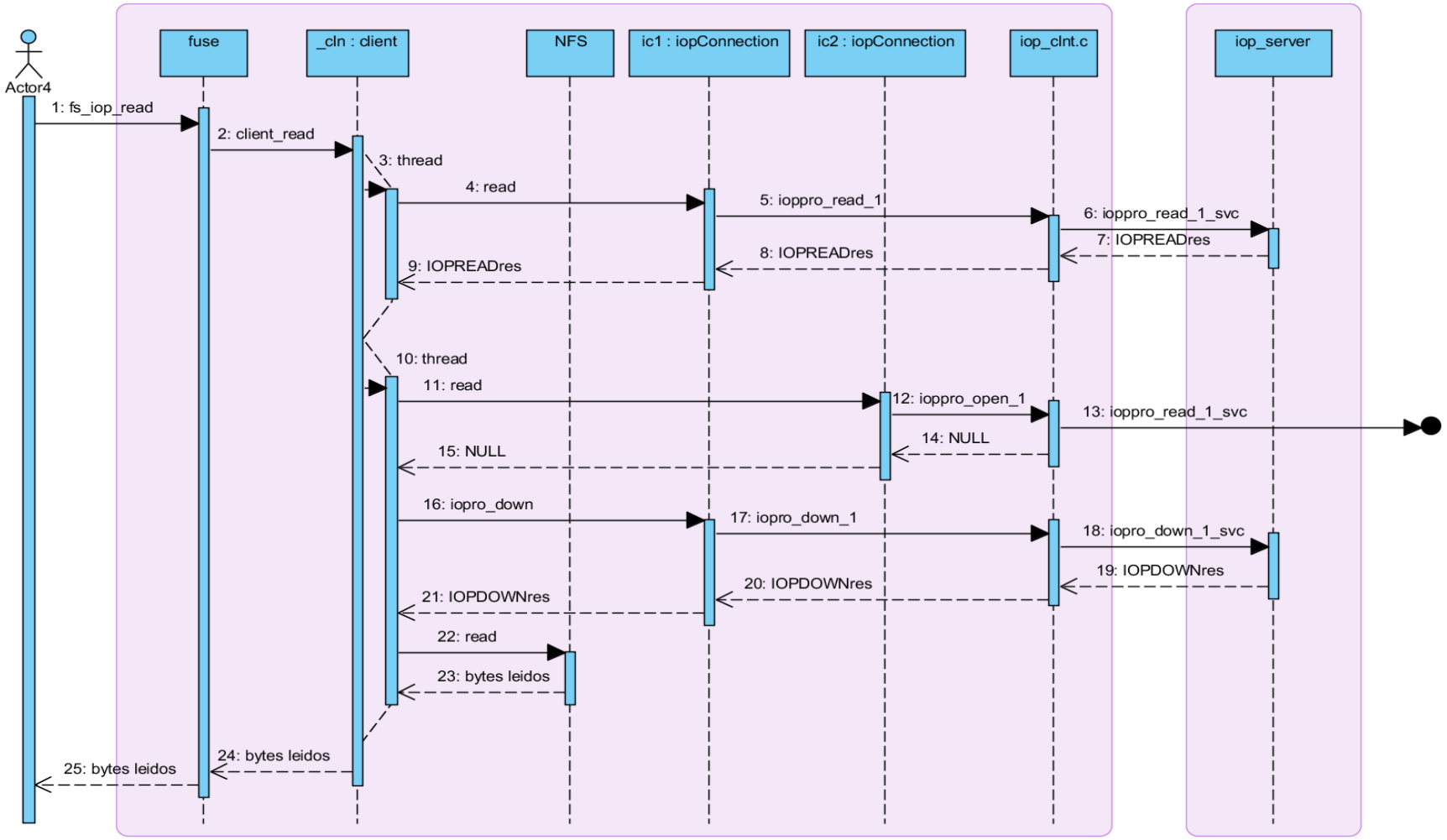


Diagrama de secuencia 5. Nodo Cliente leer

En el siguiente diagrama, se muestra como un nodo Cliente escribe dos bloques de un fichero en una red con 2 nodos IOP. El cliente crea dos procesos ligeros que se encargan de escribir los bloques necesarios en cada nodo de la red de IOP. Como el segundo proceso ligero no puede contactar con el nodo IOP se considera caído y se notifica a los otros nodos activos de la red. El bloque que debía ser escrito en el nodo IOP caído se escribe directamente de disco.

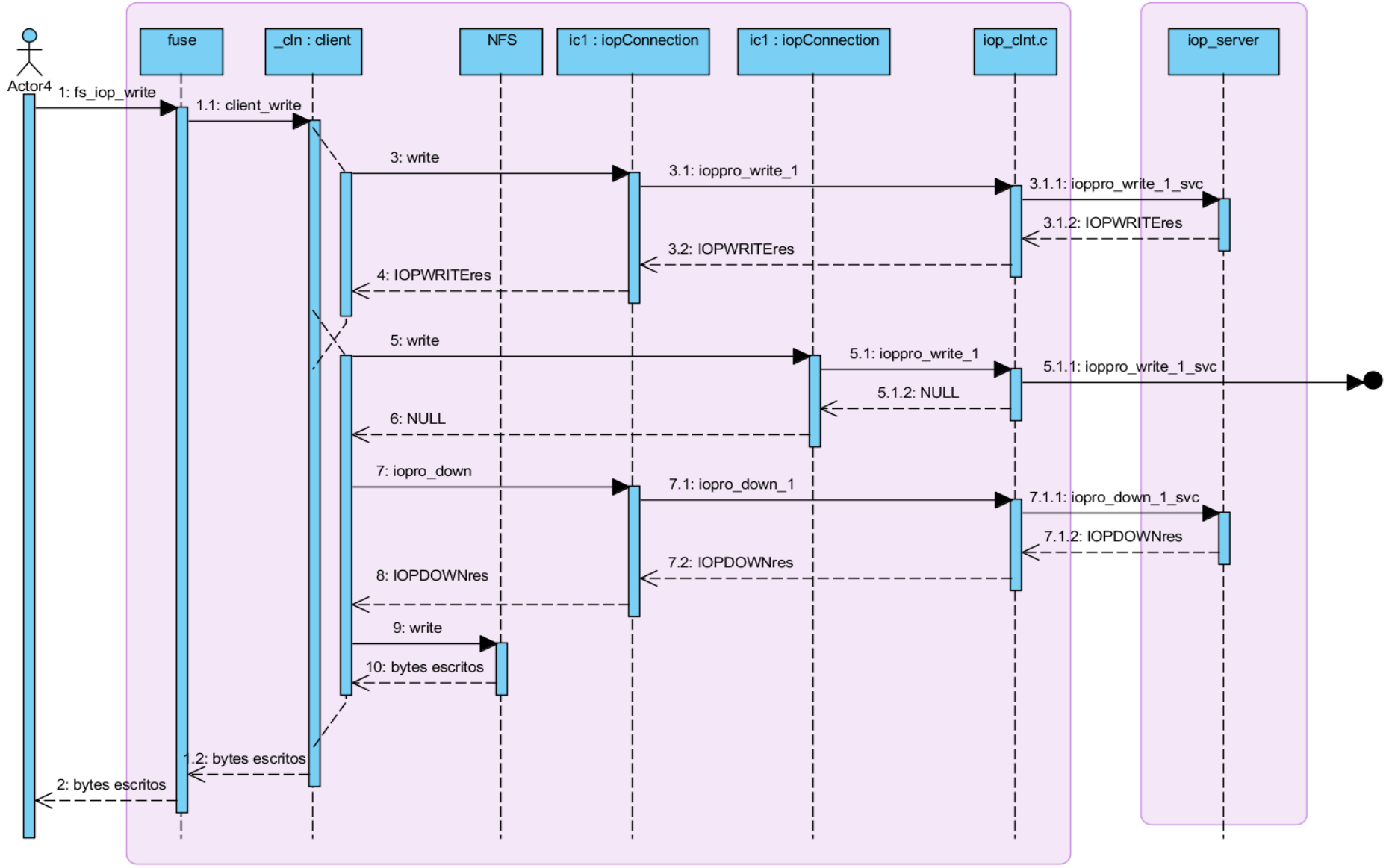


Diagrama de secuencia 6. Nodo Cliente escribir

4.5.2 Nodo de Metadatos

En el siguiente diagrama se muestra como se inicia el nodo de Metadatos.

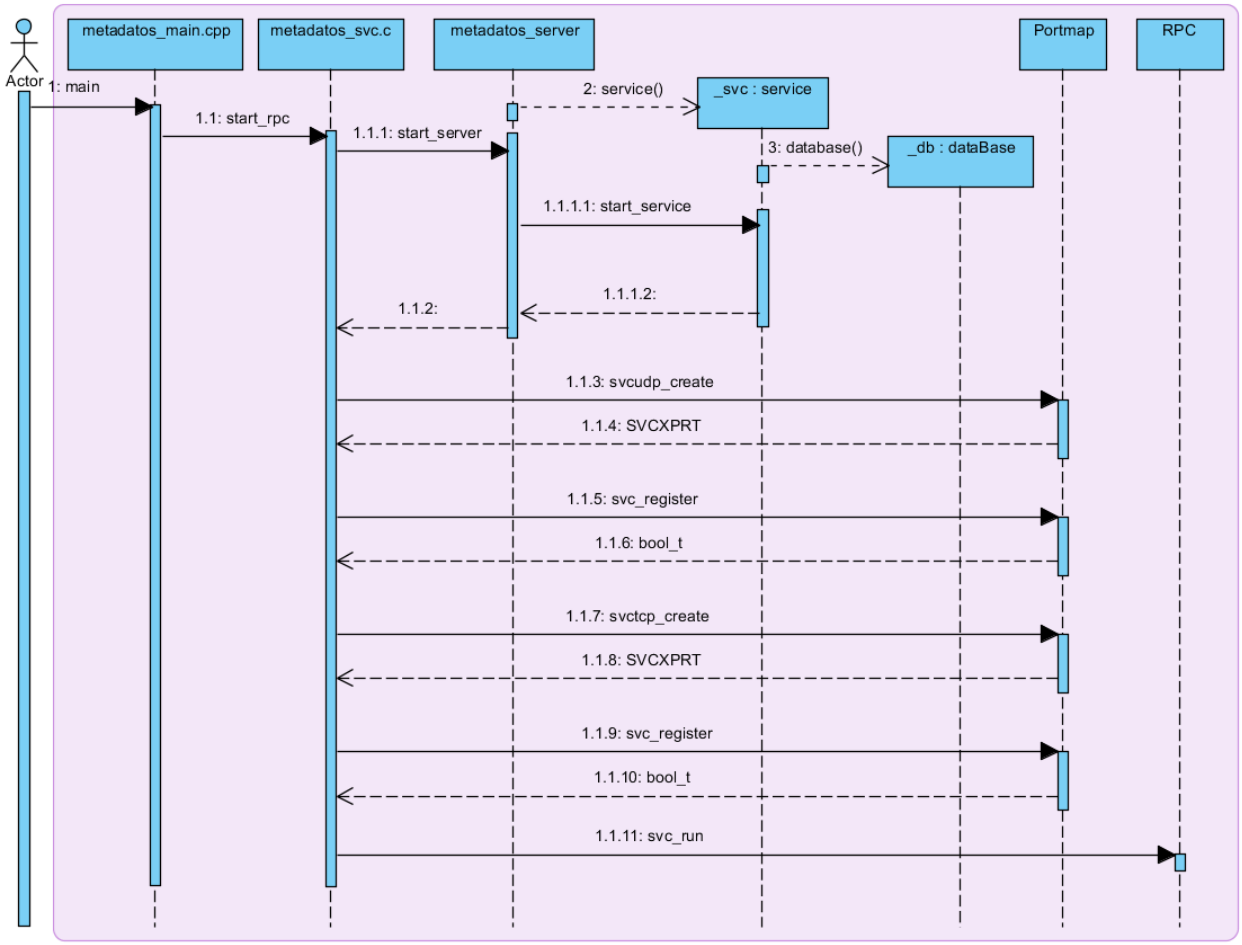


Diagrama de secuencia 7. Inicio Nodo de Metadatos

4.5.3 Nodo IOP

En el siguiente diagrama se muestra como se inicia el nodo Cliente.

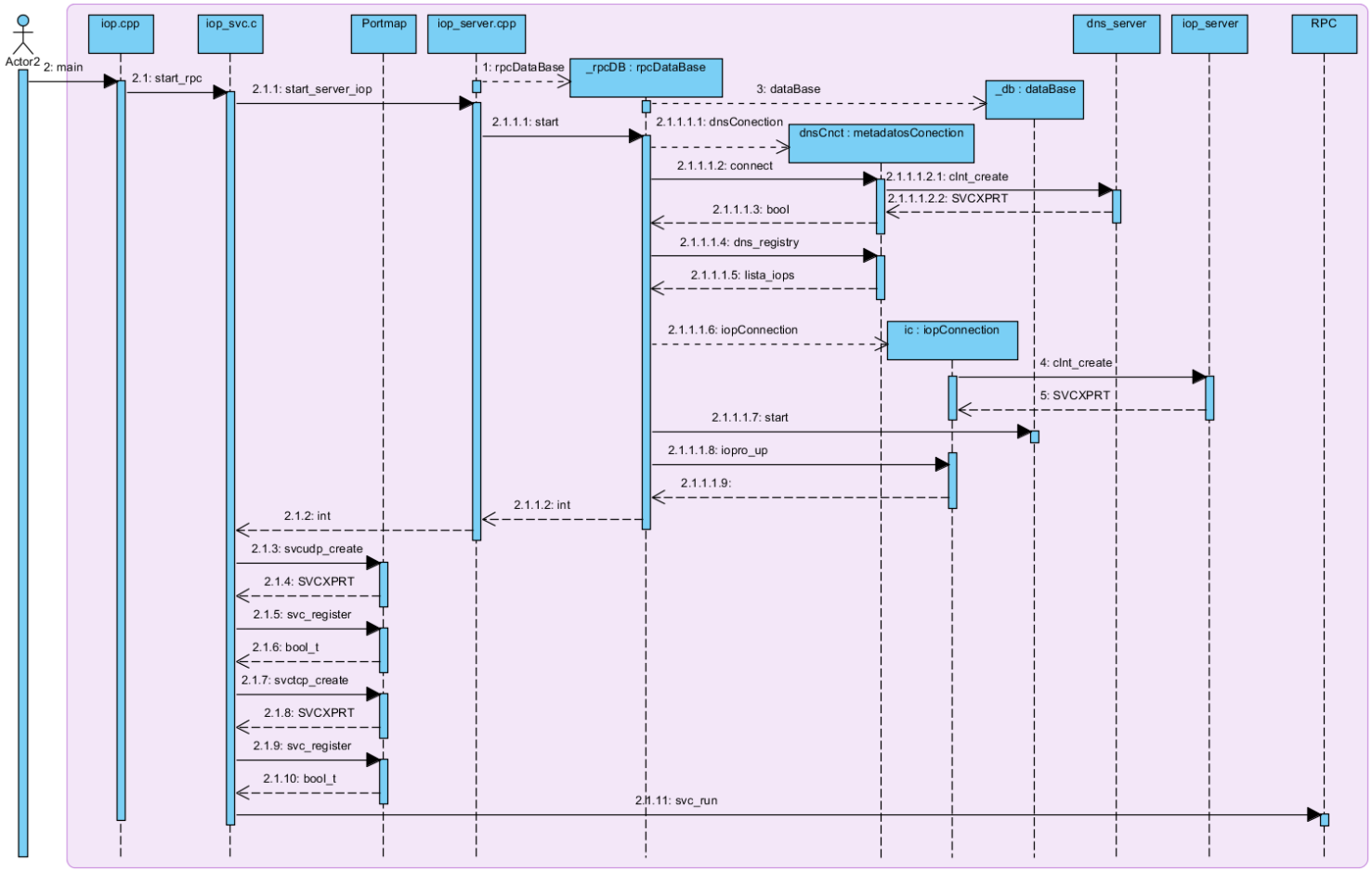


Diagrama de secuencia 8. Nodo IOP start

En el siguiente diagrama se muestra como un nodo IOP responde a una petición de lectura de un bloque. En el primer caso el bloque no está almacenado en la base de datos por lo que debe ser leído de disco y posteriormente almacenado, en el segundo caso el bloque ya se encontraba almacenado en la base de datos. Las políticas de lectura se explican en los diagramas siguientes.

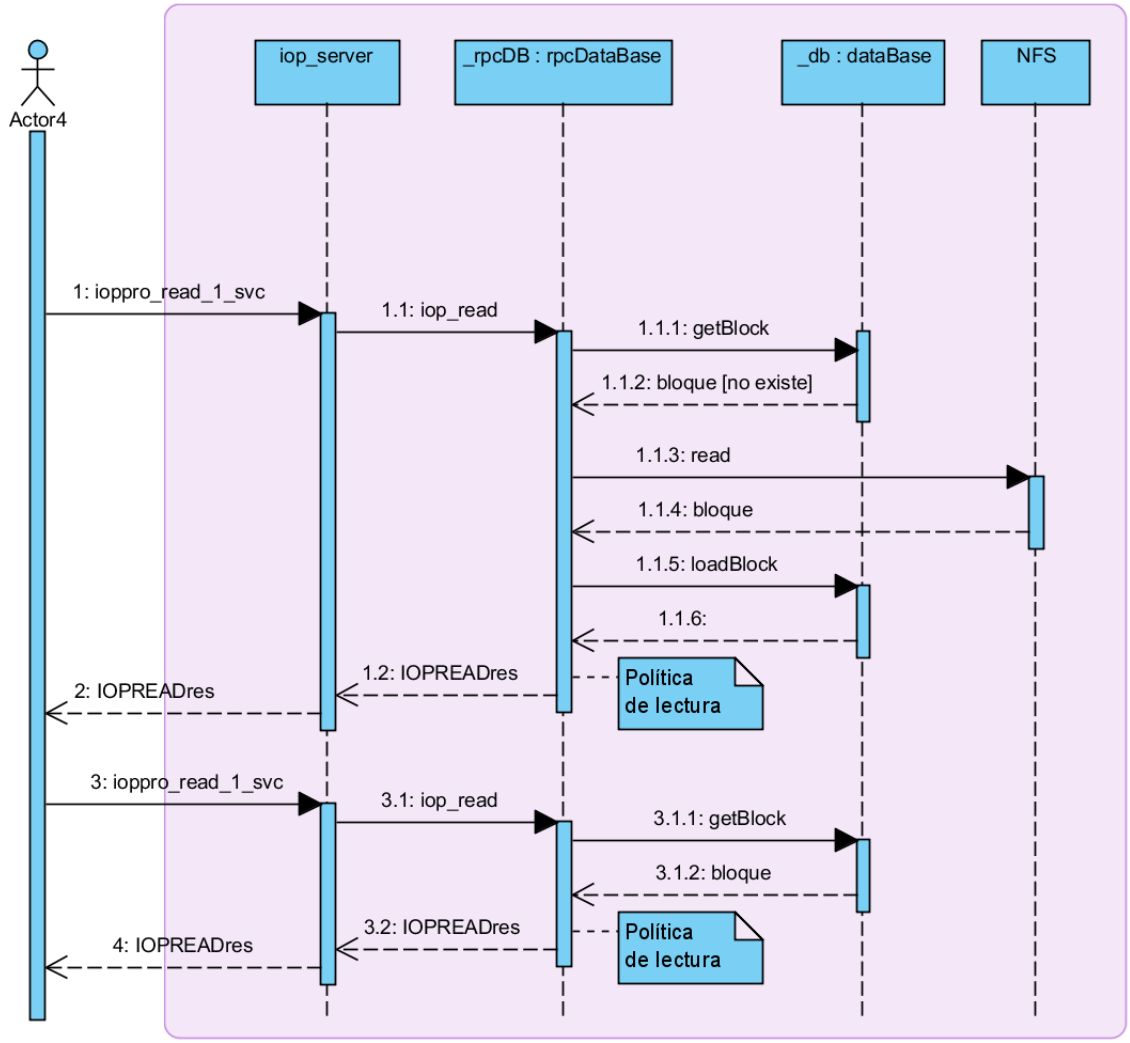


Diagrama de secuencia 9. Nodo IOP leer bloque

En el siguiente diagrama se muestra la política de lectura pasiva. Antes de devolver el bloque solicitado en una petición de lectura de bloque se procede a cargar en la base de datos los siguientes bloques que le corresponderían leer al nodo IOP. Estos bloques se leen de disco y se almacenan en la base de datos, cuando se terminar se devuelve el bloque solicitado.

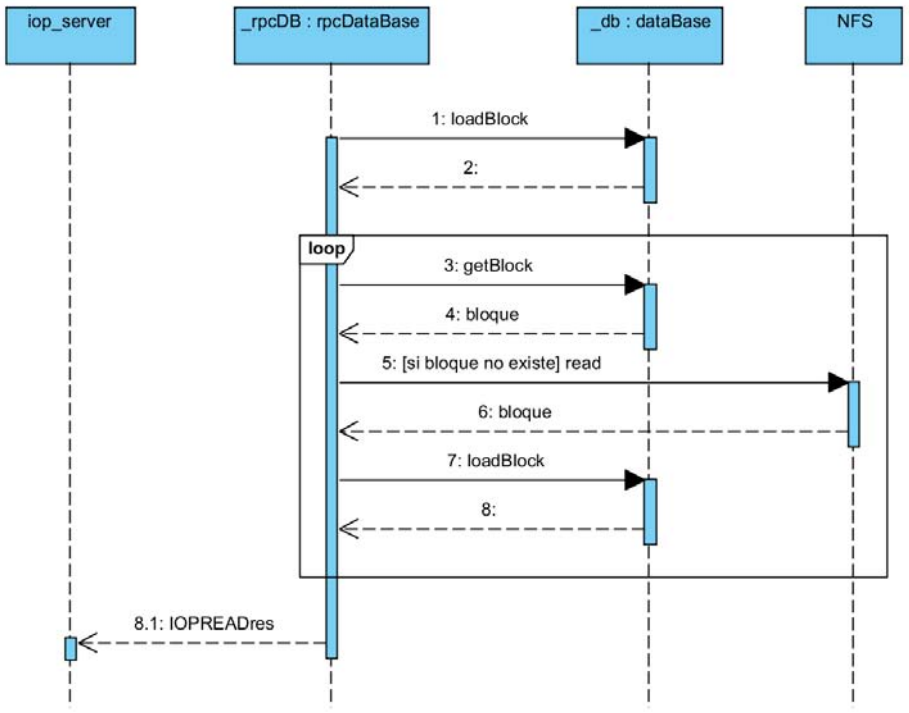


Diagrama de secuencia 10. Nodo IOP política lectura pasiva

En el siguiente diagrama se muestra la política de lectura en background. Antes de devolver el bloque solicitado en una petición de lectura de bloque se crea un proceso ligero que se encarga de cargar en la base de datos los siguientes bloques que le corresponderían leer al nodo IOP. Estos bloques se leen de disco y se almacenan en la base de datos. Se devuelve el bloque solicitado sin esperar a que termine el proceso ligero de leer los datos siguientes.

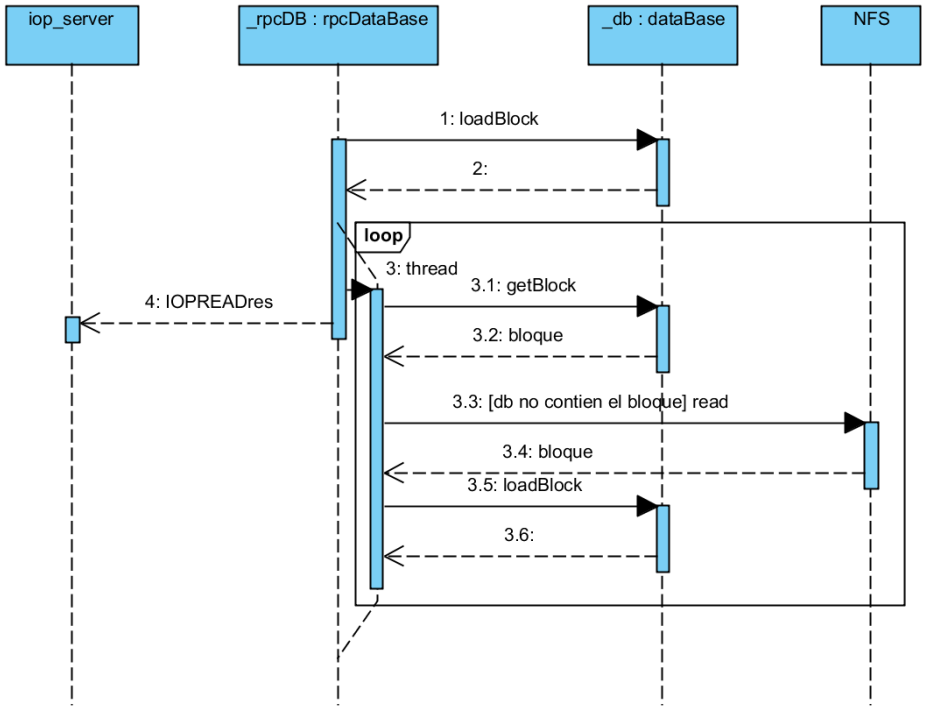


Diagrama de secuencia 11. Nodo IOP política lectura segundo plano

En el siguiente diagrama se muestra como un nodo IOP responde a una petición de escritura de un bloque. En el primer caso el bloque no está almacenado en la base de datos por lo que debe ser leído de disco y posteriormente almacenado en la base de datos con los nuevos datos escritos, en el segundo caso el bloque ya se encontraba almacenado en la base de datos por lo que se actualizan los datos. Las políticas de escritura se explican en los diagramas siguientes.

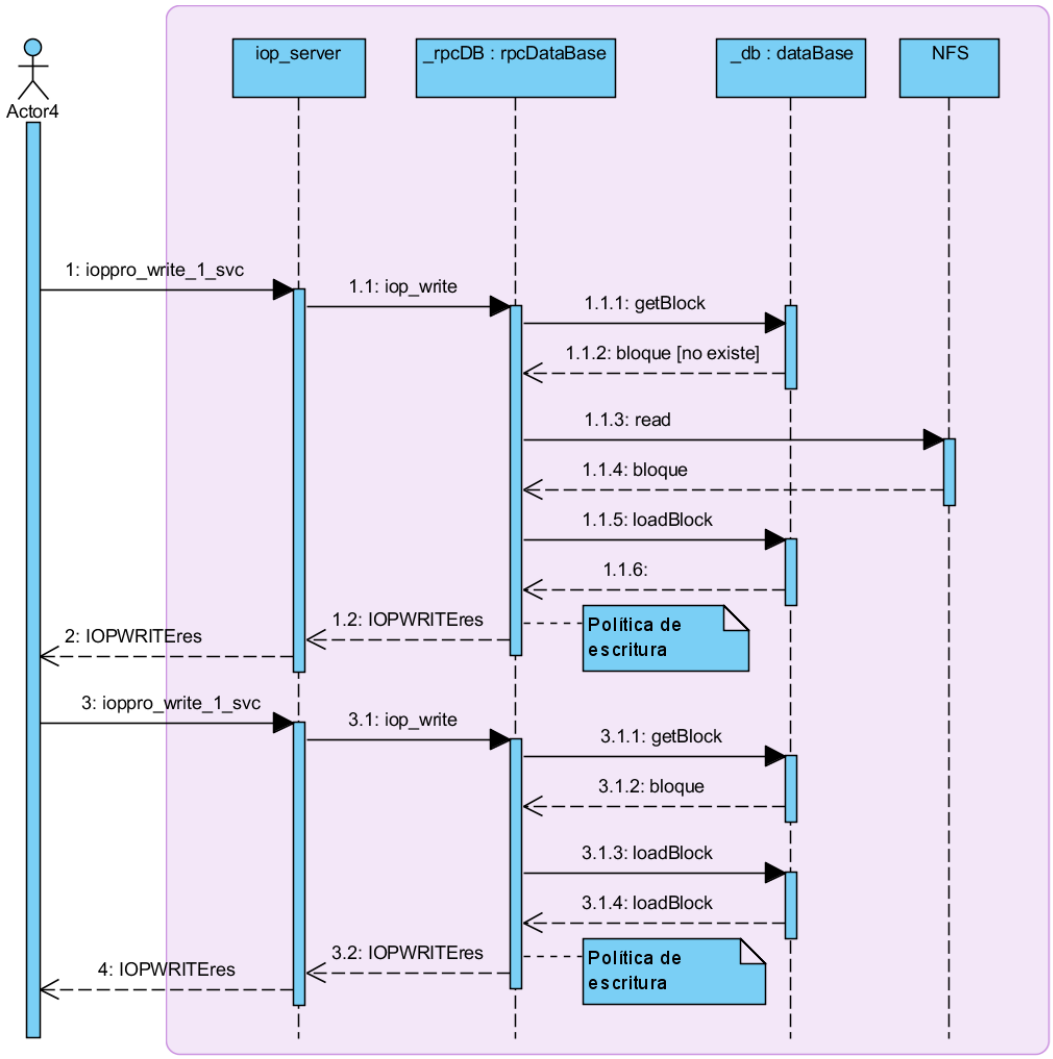


Diagrama de secuencia 12. Nodo IOP escritura

En el siguiente diagrama se muestra la política de escritura instantánea. Antes de finalizar una llamada de escritura se escriben los nuevos datos a disco, el bloque almacenado en la base de datos esta siempre limpio.

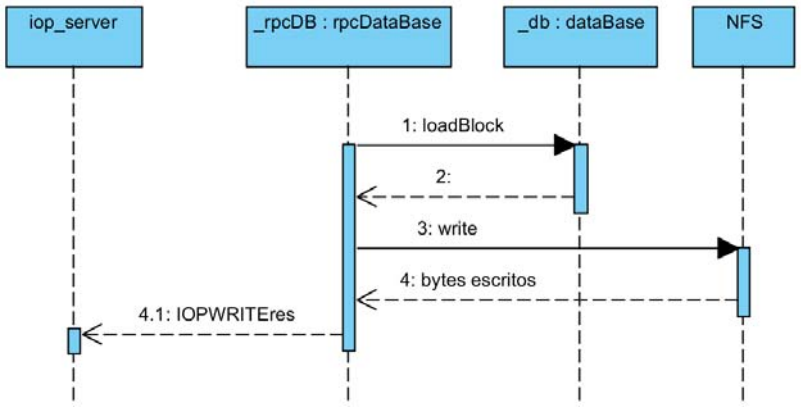


Diagrama de secuencia 13. Nodo IOP política de escritura instantánea

En el siguiente diagrama se muestra la política de escritura en N nodos IOP. Antes de finalizar una llamada de escritura se escriben los nuevos datos en N nodos IOP vecinos antes de finalizar la llamada de escritura

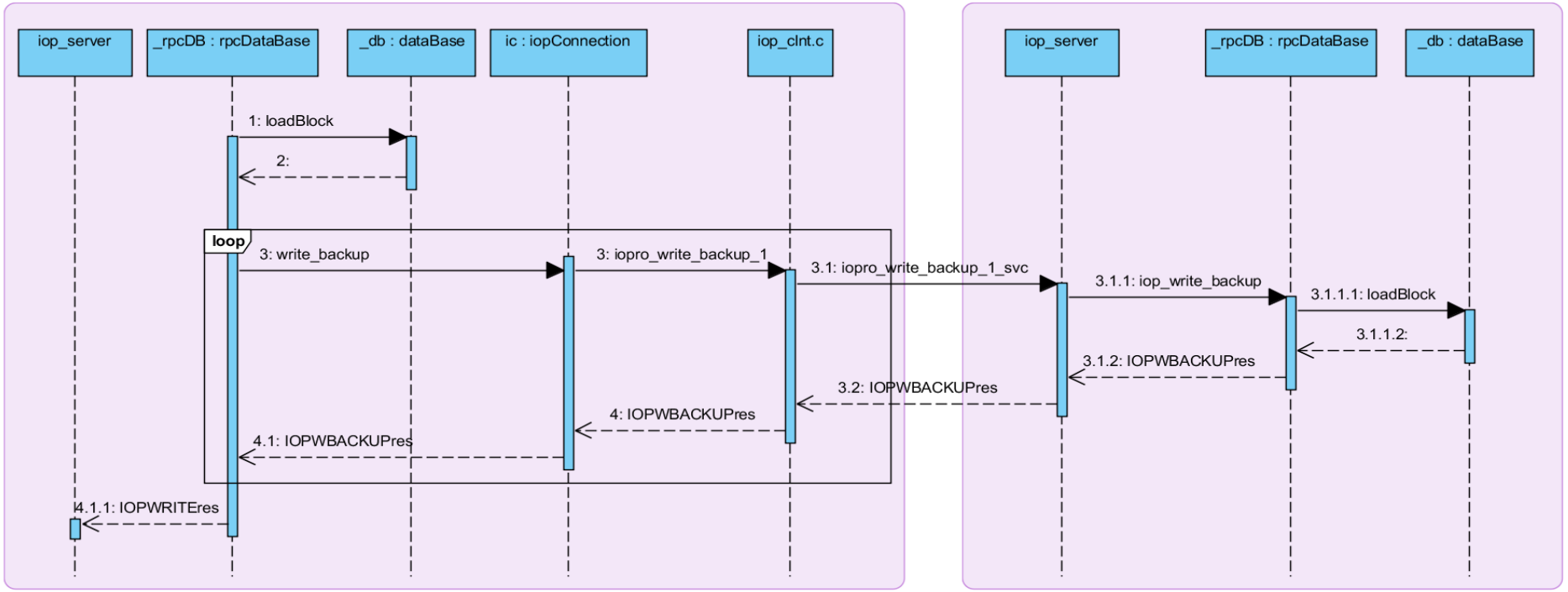


Diagrama de secuencia 14. Nodo IOP política de escritura n_iop

5. Evaluación

Durante el siguiente capítulo se expone como se ha llevado a cabo la evaluación del sistema de ficheros. Se definirá el entorno donde se han realizado las pruebas, las pruebas elegidas para evaluar el sistema, el propósito de dichas pruebas, los resultados obtenidos en las pruebas y finalmente se concluirá con un resumen final de los resultados obtenidos.

5.1 Entorno de las pruebas

La plataforma utilizada para la evaluación de las pruebas ha sido un cluster de 16 computadores. Las características de los computadores se describen a continuación:

- **CPU**: Quadcore Intel(R) Xeon(R) E5405 @ 2.00GHz
- **Memoria Ram**: 4 GB
- Red: 1000 Mbits (Gigaethernet)

5.2 Definición de las pruebas

Se realizaran 90 pruebas, que se agruparan en los 5 conjuntos siguientes:

- 1 nodo cliente leyendo un fichero de 20Mb.
- 4 nodos cliente leyendo 4 ficheros diferentes de 20Mb.
- 4 nodos cliente leyendo un mismo fichero de 20Mb.

- 8 nodos cliente leyendo 8 ficheros diferentes de 20Mb.
- 8 nodos cliente leyendo un mismo fichero de 20Mb.

Cada uno de estos 5 conjuntos se divide en 9 subconjuntos:

A. 1 nodo IOP con tamaño de bloque 64Kb
B. 1 nodo IOP con tamaño de bloque 128Kb
C. 1 nodo IOP con tamaño de bloque 256Kb
D. 4 nodo IOP con tamaño de bloque 64Kb
E. 4 nodo IOP con tamaño de bloque 128Kb
F. 4 nodo IOP con tamaño de bloque 256Kb
G. 8 nodo IOP con tamaño de bloque 64Kb
H. 8 nodo IOP con tamaño de bloque 128Kb
I. 8 nodo IOP con tamaño de bloque 256Kb

Cada uno de estos subconjuntos tienen 2 pruebas, una primera lectura con la cache a 0%, es decir no se ha precargado ningún dato y una segunda lectura con la cache al 100%, que significa que los datos ya han sido leídos con anterioridad.

5.3 Propósito de las pruebas

El propósito de las pruebas es comparar el tiempo que tardan en ejecutarse dichas pruebas sobre un sistema de ficheros NFS y sobre el sistema de ficheros implementado.

Se pretenden analizar diversas situaciones que hagan ver las virtudes y carencias del nuevo sistema de ficheros, frente al sistema de ficheros NFS.

5.4 Lecturas

En esta sección se muestran los resultados obtenidos ejecutando operaciones de lectura sobre el sistema de ficheros y comparándolo con NFS, se analizan diferentes situaciones y diferentes configuraciones. Posteriormente se analizaran y comentaran los resultados obtenidos en las diferentes pruebas.

En la graficas se representara mediante 2 líneas horizontales los tiempos de lectura sobre NFS, la línea superior representa el tiempo de una primera lectura y la línea inferior el tiempo de una segunda lectura, con esto se consigue ver el uso de la cache. En la grafica se muestran los tiempos obtenidos realizando la misma prueba sobre el sistema de ficheros diseñado con diferentes configuraciones. Las distintas

configuraciones de las 9 pruebas de cada grafica se detallan en la tabla debajo de cada una. Las variaciones de las pruebas son el número de IOPs que forman la red y el tamaño de bloque que se almacena en la base de datos. Cada prueba de la A a la I contiene dos mediciones una con la cache al 0% y otra con la cache al 100% (barra azul y roja respectivamente).

5.4.1 Pruebas con 1 cliente

A continuación se muestran los resultados obtenidos con 1 solo cliente leyendo un fichero de 20Mb en peticiones de 128Kb.

En la configuración con 1 solo IOP, los valores se asemejan a los de NFS en la primera lectura del fichero y son ligeramente superiores cuando el fichero ya se encuentra en cache.

En la configuración con 4 IOPs, los valores de la primera lectura del fichero son mejores que los obtenidos en NFS, en especial cuando el tamaño de bloque es igual o superior al tamaño de las peticiones de lectura.

En la configuración con 8 IOPs, es similar al caso de 4 IOPs salvo en el caso de la primera lectura cuando el tamaño de bloque es la mitad que el de las peticiones de lectura.

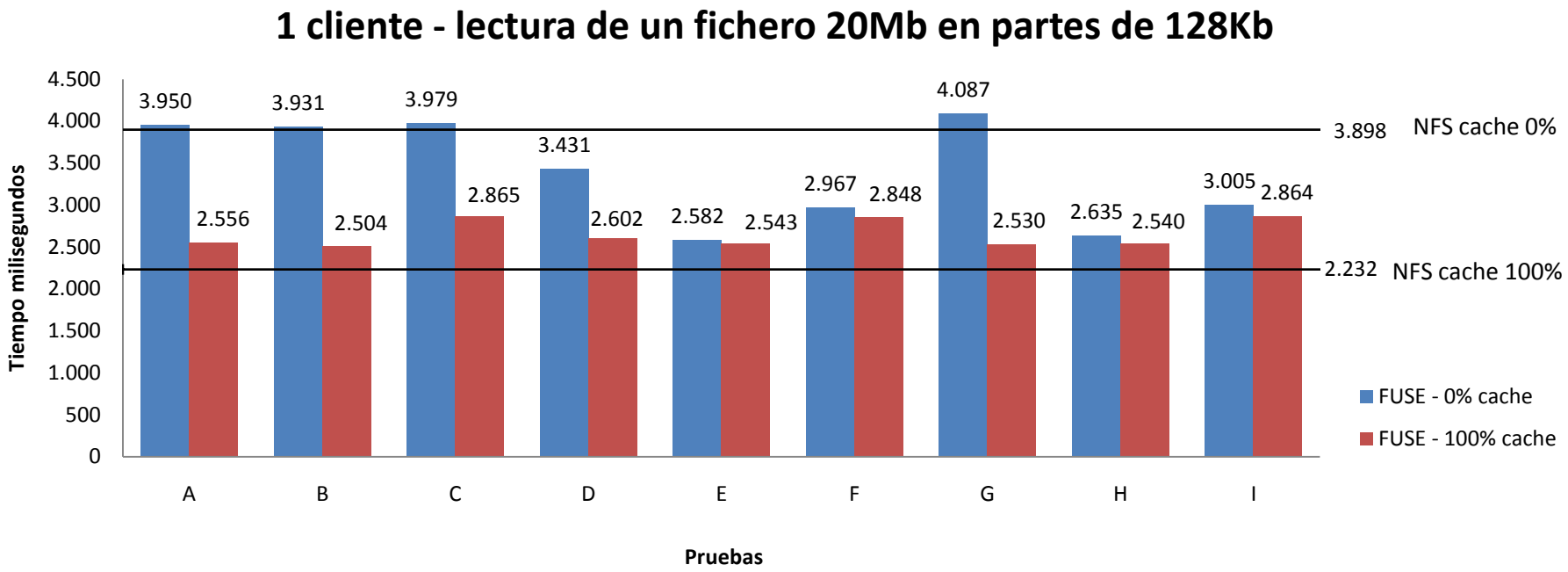


Gráfico 1. Prueba con 1 cliente y 1 fichero

	NFS	A	B	C	D	E	F	G	H	I
Cache 0% (tiempo en milisegundos)	3.898	3.950	3.931	3.979	3.431	2.582	2.967	4.087	2.635	3.005
Cache 100% (tiempo en milisegundos)	2.232	2.556	2.504	2.602	2.602	2.543	2.848	2.530	2.540	2.864
Numero IOPs	-	1	1	1	4	4	4	8	8	8
Tamaño bloque (Kb)	-	65	128	256	65	128	256	65	128	256

Tabla 1. Prueba con 1 cliente y 1 fichero

5.4.2 Pruebas con 4 clientes sobre 4 ficheros diferentes

A continuación se muestran los resultados obtenidos con 4 clientes leyendo 4 fichero diferentes de 20 Mb en peticiones de 128 Kb.

Los datos obtenidos en esta prueba se asemejan a los obtenidos sobre NFS, salvo el caso de un solo nodo IOP y tamaño de bloque es el doble que el tamaño de las peticiones de lectura, en el que se penaliza que las peticiones de los clientes converjan sobre un mismo IOP. Esto se debe a que un mismo IOP contiene los datos de 2 lecturas consecutivas.

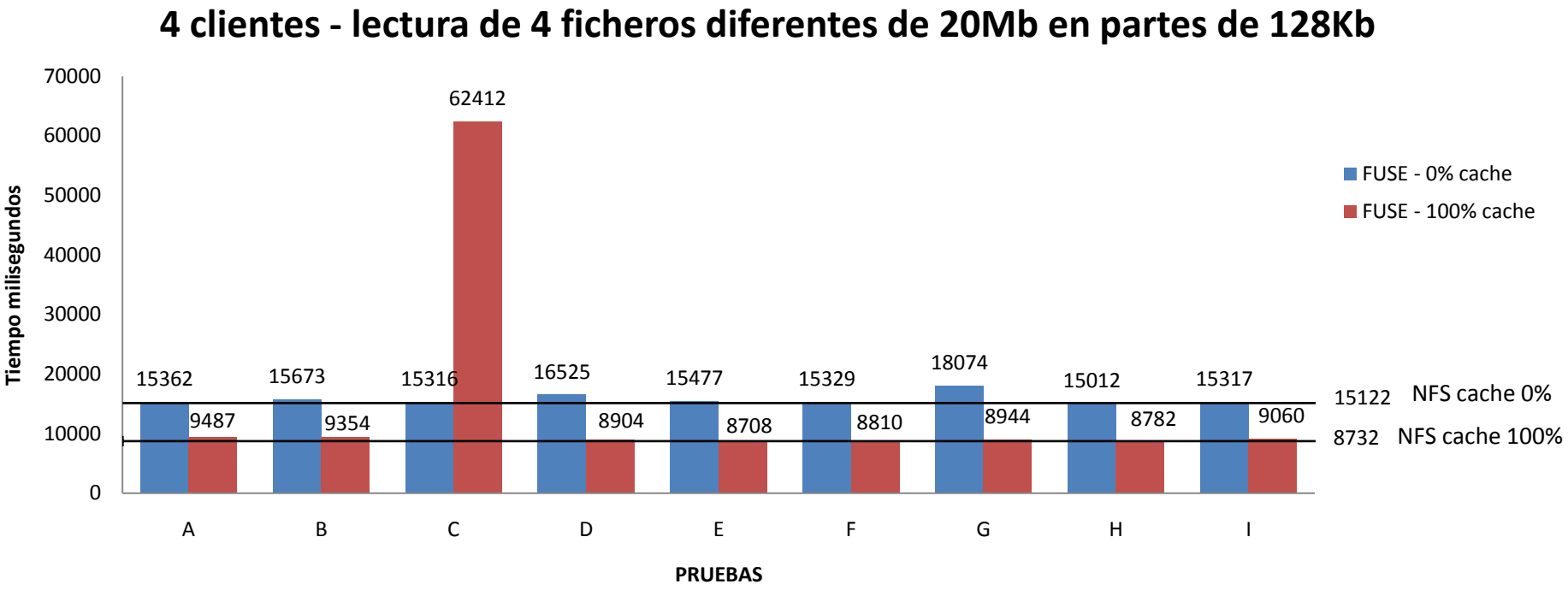


Gráfico 2. Prueba con 4 clientes y 4 ficheros

	NFS	A	B	C	D	E	F	G	H	I
Cache 0% (tiempo en milisegundos)	15.122	15.362	15.673	15.316	16.525	15.477	15.329	18.074	15.012	15.317
Cache 100% (tiempo en milisegundos)	8.732	9.487	9.354	62412	8.904	8.708	8.810	8.944	8.782	9.060
Numero IOPs	-	1	1	1	4	4	4	8	8	8
Tamaño bloque (Kb)	-	65	128	256	65	128	256	65	128	256

Tabla 2. Prueba con 4 clientes y 4 ficheros

5.4.3 Pruebas con 4 clientes sobre un mismo fichero

A continuación se muestran los resultados obtenidos con 4 clientes leyendo un mismo fichero de 20Mb en peticiones de 128Kb.

Las primeras lecturas son entorno a un 20% - 30% más rápido que los datos obtenidos sobre NFS.

Los datos obtenidos en las segundas lecturas son similares a los obtenidos sobre NFS, salvo en los casos en los que el tamaño de bloque es el doble que el tamaño de las peticiones de lectura, en el que se penaliza que las peticiones de los clientes converjan sobre un mismo IOP. Esto se debe a que un mismo IOP contiene los datos de 2 lecturas consecutivas.

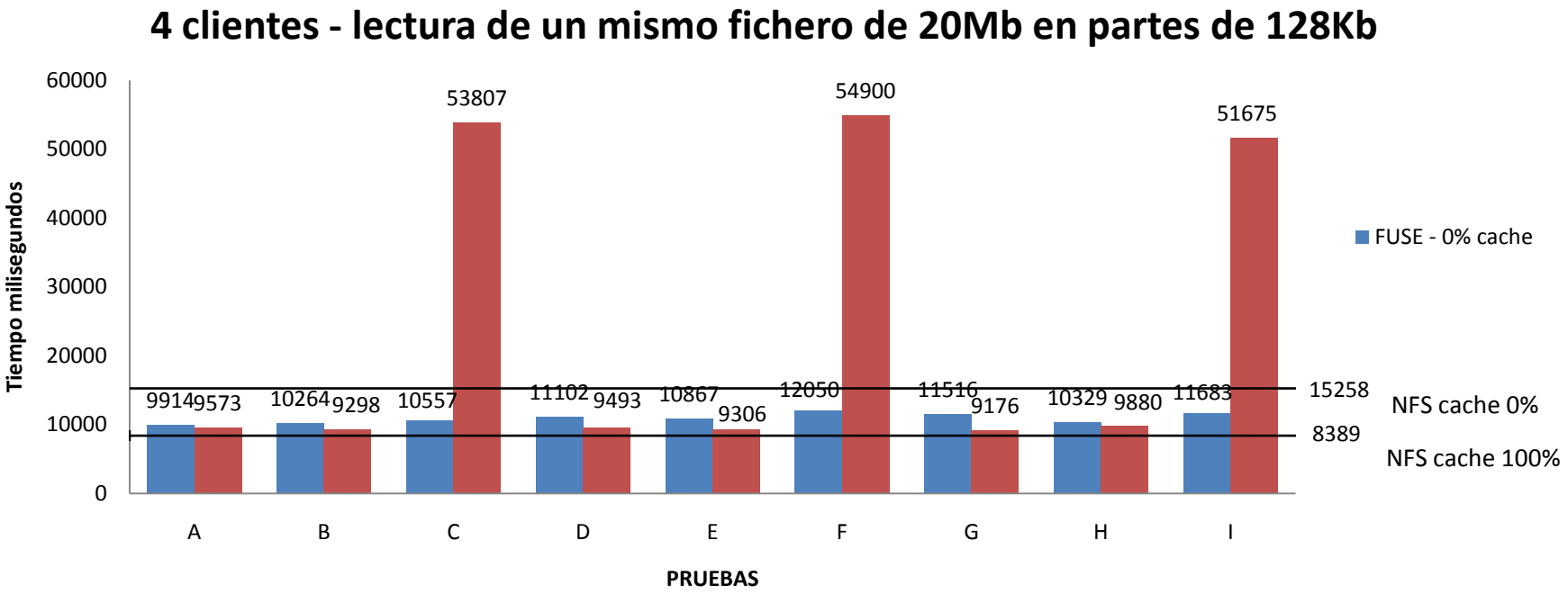


Gráfico 1. Prueba con 4 clientes y 4 ficheros

	NFS	A	B	C	D	E	F	G	H	I
Cache 0% (tiempo en milisegundos)	15.258	9.914	10.264	10.557	11.102	10.867	12.050	11.516	10.329	11.683
Cache 100% (tiempo en milisegundos)	8.389	9.573	9.298	53807	9.493	9.306	54.900	9.176	9.880	51.675
Numero IOPs	-	1	1	1	4	4	4	8	8	8
Tamaño bloque (Kb)	-	65	128	256	65	128	256	65	128	256

Tabla 1. Prueba con 4 clientes y 4 ficheros

5.4.4 Pruebas con 8 clientes sobre 8 ficheros diferentes

A continuación se muestran los resultados obtenidos con 8 clientes leyendo 8 ficheros diferentes de 20Mb en peticiones de 128Kb.

Los tiempos son similares a los de NFS, salvo en las segundas lecturas con un solo nodo IOP y en los casos en los que el tamaño de bloque es el doble que el tamaño de las peticiones de lectura.

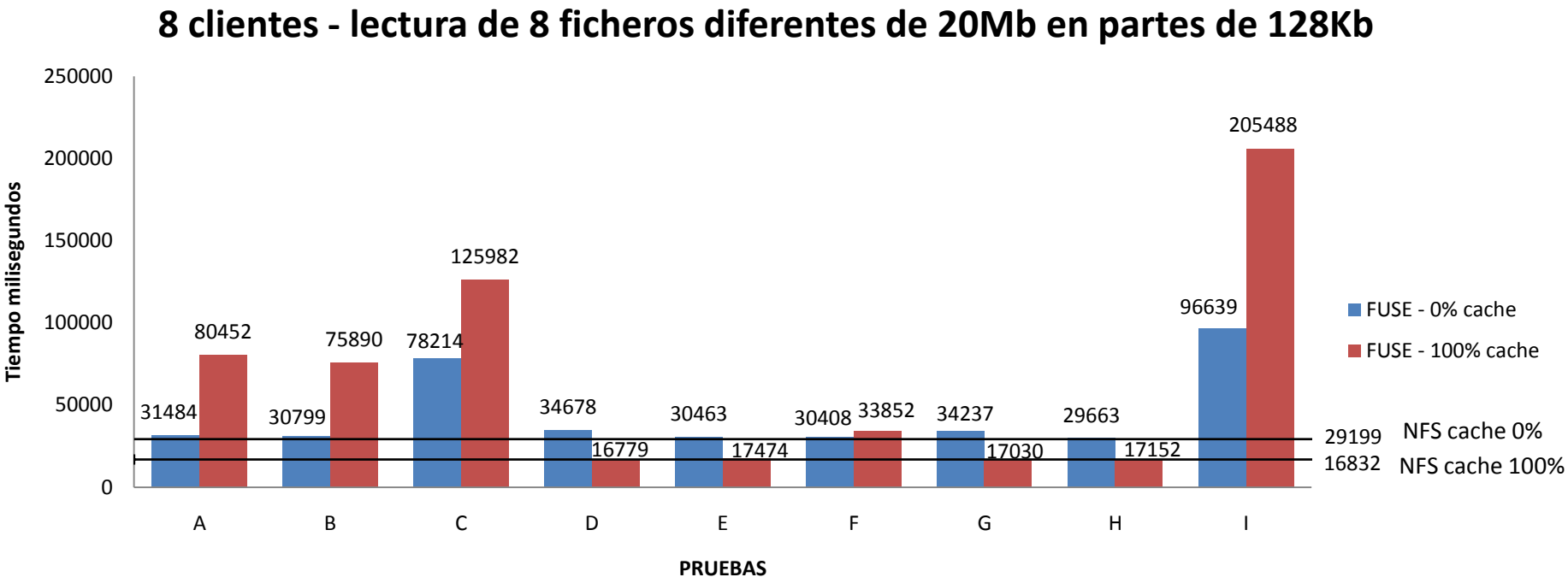


Gráfico 2. Prueba 8 clientes y 8 fichero

	NFS	A	B	C	D	E	F	G	H	I
Cache 0% (tiempo en milisegundos)	29.199	31.484	30.799	78.214	34.678	30.463	30.408	34.237	29.663	96.639
Cache 100% (tiempo en milisegundos)	16.832	80.452	75.890	125982	16.779	17.474	33.852	17.030	17.152	205.488
Numero IOPs	-	1	1	1	4	4	4	8	8	8
Tamaño bloque (Kb)	-	65	128	256	65	128	256	65	128	256

Tabla 2. Prueba 8 clientes y 8 fichero

5.4.5 Pruebas con 8 clientes sobre un mismo fichero

A continuación se muestran los resultados obtenidos con 8 clientes leyendo 8 ficheros diferentes de 20Mb en peticiones de 128Kb.

Los tiempos de primeras lecturas son entorno a un 30% mejores que los de NFS, salvo los casos en los que el tamaño de bloque es el doble que el tamaño de las peticiones de lectura.

Los tiempos de segundas lecturas son peores que los de NFS, salvo en el caso de la prueba D, en la que la convergencia de llamadas sobre un mismo IOP es salvada gracias que los bloques se encuentran más repartidos entre los nodos IOP debido a que una petición de lectura se encuentra en 2 nodos IOP distintos.

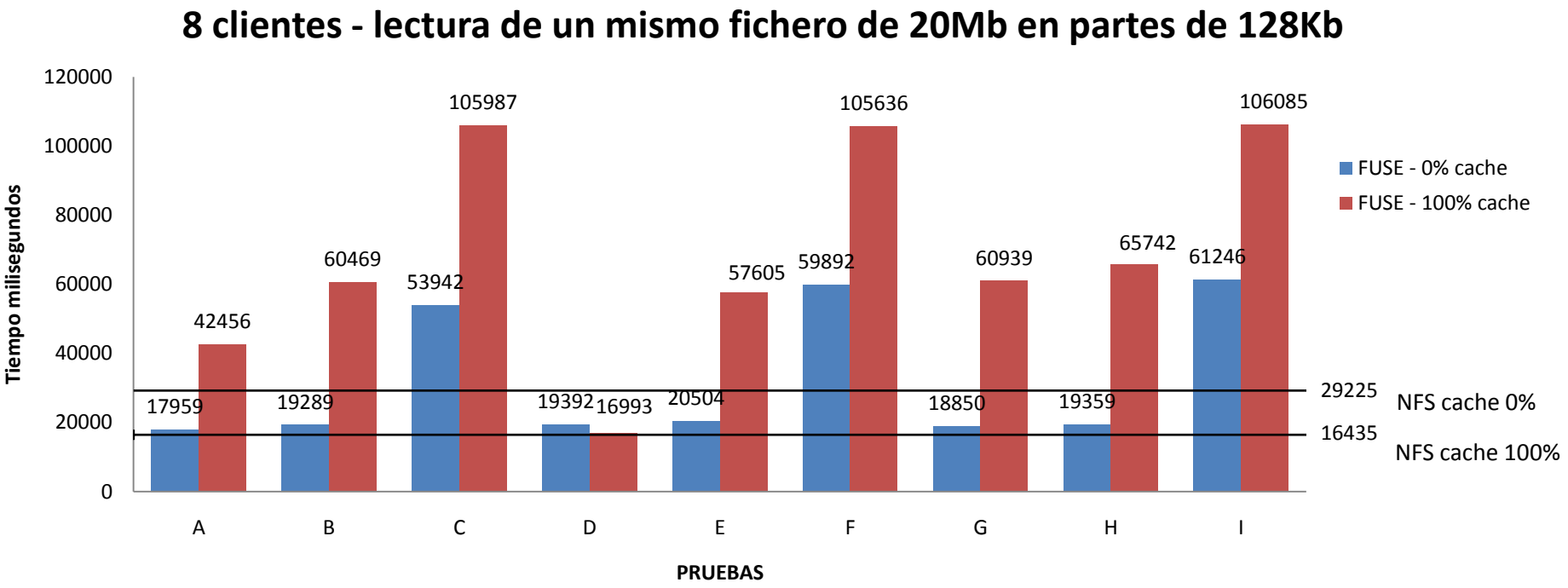


Gráfico 3. Prueba 8 clientes y 1 fichero

	NFS	A	B	C	D	E	F	G	H	I
Cache 0% (tiempo en milisegundos)	29.225	17.959	19.289	53.942	19.392	20.504	59.892	18.850	19.359	61.246
Cache 100% (tiempo en milisegundos)	16.435	42.456	60.469	105987	16.993	57.605	105.636	60.939	65.742	61.246
Numero IOPs	-	1	1	1	4	4	4	8	8	8
Tamaño bloque (Kb)	-	65	128	256	65	128	256	65	128	256

Tabla 3. Prueba 8 clientes y 1 fichero

5.5 Resumen

El sistema presenta tres grandes carencias o problemas, el primer problema se centra en la elección del protocolo de comunicación, que no permite realizar varias llamadas iguales de forma paralela. Esto penaliza a los clientes que no pueden realizar llamadas paralelas a varios nodos IOP de forma simultánea.

El segundo es que los nodos IOP no pueden atender a varias llamadas simultáneamente, esto penaliza enormemente la resolución de llamadas por parte de los clientes.

El tercer gran problema es la base de datos, que cuando contiene un cierto número de bloques las peticiones de búsqueda de bloques tardan demasiado. Esto penaliza la resolución de llamadas por parte de los nodos IOP.

A pesar de estos 3 grandes problemas se han conseguidos valores muy satisfactorios en algunas pruebas. Esto hace prever que cuando se solucionen dichos problemas, el sistema de ficheros ofrecerá un rendimiento mucho mayor.

6. Conclusiones y trabajos futuros

En este capítulo se revisan los objetivos planteados que fueron marcados al inicio del proyecto, así como del grado de cumplimiento de los mismos. Por último se proponen futuras mejoras y líneas de trabajo.

6.1 Conclusiones

En la sección 2.3 se definieron seis objetivos para el presente proyecto fin de carrera:

- **Estudiar otros sistemas de ficheros distribuidos/paralelos como GFS**.

  Se han estudiado diversos sistemas de ficheros, su funcionamiento y arquitectura. Se han extraído ideas de ellos, para la realización del nuevo sistema de ficheros.

- **Estudiar el protocolo de comunicación RPC (*Remote Procedure Call*).**

  Se ha estudiado el protocolo de comunicación y se ha integrado en la implementación del sistema de ficheros.

- **Estudiar la base de datos BerkeleyDB con almacenamiento de datos en memoria y en disco.**

  Se ha estudiado la documentación de la base de datos BerkelyDB, así como diversos ejemplos de uso. Se ha integrado en la implementación del

sistema de ficheros, almacenando la base de datos tanto en disco como en memoria.

- **Estudiar el sistema de ficheros FUSE (*Filesystem in Userspace*), y aplicaciones realizadas usando este sistema.**

  Se han probado diversos proyectos que hacían uso de la librería FUSE y se han estudiado. Se ha integrado en el sistema de ficheros.

- **Desarrollar un sistema de ficheros que reduzca los cuellos de botella sobre NFS.**

  Se ha desarrollado un sistema de ficheros con tres tipos de nodos. El nodo IOP poseía tres políticas de escritura y lectura. Se implemento una única política de distribución de datos. El sistema ofrece mejoras sustanciales en el acceso a datos, con ciertas configuraciones.

- **Evaluar el funcionamiento de la aplicación.**

  Se ha analizado el rendimiento del sistema en pruebas de lectura. La opción de lectura era el principal objetivo de este proyecto, posteriormente se añadieron también las funcionalidades de escritura y trabajo con nodos caídos. Estas 2 últimas funcionalidades no han sido evaluadas en el presente proyecto, pero si ha sido verificado su correcto funcionamiento.

El sistema de fichero implementado reduce los tiempos de acceso a los datos, distribuyendo estos en diferentes nodos. Las distintas políticas de escritura y lectura permiten adaptar el sistema de ficheros a diferentes demandas. Tener los datos distribuidos reduce el tráfico de red entorno a un nodo y permite alcanzar velocidades de transferencia mayores. Almacenar los datos de en memoria reduce los tiempos de acceso frente al acceso en disco. El uso de la librería FUSE facilita la implantación del sistema de ficheros, sin repercutir en el sistema de ficheros que se pretende sustituir

Con todo esto, podemos concluir que este tipo de sistemas de almacenamiento son muy adecuados en entornos que requieren trabajar con gran volumen de información.

6.2 Trabajos futuros

Para ampliar las posibilidades que puede ofrecer este Proyecto Fin de Carrera, se proponen implementar las siguientes funcionalidades:

- Un **sistema de cache en el nodo cliente**, las lecturas de bloques sobre ficheros que están abiertos en modo lectura se almacenaran en una cache en el cliente, este reducirá las llamadas sobre los nodos IOP. Cuando un fichero es abierto en modo escritura se deberá notificar a los clientes que lo tuviesen abiertos en modo lectura que ya no pueden hacer cache de ese fichero.

- **Modificar el numero de nodos IOP en una red en funcionamientos**, los nodos IOP notifican a sus clientes cuando se añade un nuevo nodo IOP a la red o cuando se cae un nodo, esto permite la modificación de las redes en caliente y también reduce el número de llamadas a nodos caídos.

- **Nuevas políticas de escritura y lectura**, en vez de asociar una política de lectura o escritura a un nodo IOP:
  - **Asociar las políticas a los clientes**: esto permite que si un cliente va a realizar lecturas consecutivas que tenga una política de lectura NEXT_BACKGROUND, o si un cliente va a realizar lecturas aleatorias tenga una política de lectura ONLY_DEMAND.
  - **Asociar las políticas a los directorios o ficheros**: sobre directorios o ficheros de respaldo interesa que los datos se escriban directamente a disco, mientras que sobre directorios o ficheros con un gran volumen de llamadas E/S interesa mantener los datos en la cache de los nodos IOP.

- **Volcar información de un nodo IOP a disco**. Esto permite eliminar un nodo IOP de una red sin que entren en funcionamiento los métodos de control de consistencia.

- **Implementar la llamada POSIX rm y rmdir**. Elimina de las bases de datos de los nodos IOP todo los ficheros referentes a estas 2 llamadas.

- **Implementar la llamada POSIX rename**. Esto permite actualizar los bloques en la base de datos sin necesidad de volverlos a cargar.

- **Paralelizar las llamadas de escritura y lectura en los nodos IOP**, esto evitara los cuellos de botella producidos por acceder varios clientes a un mismo nodo.

- **Analizar el rendimiento de las escrituras.** Realizar pruebas para medir el rendimiento en procesos de escritura y procesos mixtos de escritura y lectura.

    - **Pruebas de escritura:**
        - 1 nodo cliente leyendo un fichero de 20Mb.
        - 4 nodos cliente leyendo 4 ficheros diferentes de 20Mb.
        - 4 nodos cliente leyendo un mismo fichero de 20Mb.
        - 8 nodos cliente leyendo 8 ficheros diferentes de 20Mb.
        - 8 nodos cliente leyendo un mismo fichero de 20Mb.

      Con las diferentes variaciones (A-I) detallas en la sección 5.2

    - **Procesos mixtos de escritura y lectura:**
        - 2 nodos cliente leyendo 2 ficheros diferentes de 20Mb y 2 nodos escribiendo en los 2 ficheros.
        - 2 nodos cliente leyendo un mismo fichero de 20Mb y 2 nodos escribiendo en ese mismo fichero.
        - 4 nodos cliente leyendo 4 ficheros diferentes de 20Mb y 4 nodos escribiendo en los 4 ficheros.
        - 4 nodos cliente leyendo un mismo fichero de 20Mb y 4 nodos escribiendo en ese mismo fichero.

      Con las diferentes variaciones (A-I) detallas en la sección 5.2

- **Analizar el rendimiento cuando se caen nodos de una red.** Realizar pruebas de escritura, lectura y mixtas sobre redes con nodos IOP caídos, y medir el rendimiento del sistema. Repetir las pruebas detallas anteriormente con las siguientes variaciones:
    - 25% de los nodos caídos.
    - 50% de los nodos caídos.

- **Peticiones paralelas en los nodos IOP.** Desarrollar que los nodos IOP puedan responder varias peticiones simultáneamente, al menos en las peticiones de lectura.

- **Mejorar el rendimiento de la base de datos.**

7. Planificación y presupuesto

Durante el siguiente apartado se mostrara de forma gráfica en la planificación del proyecto así como todos los aspectos concernientes al presupuesto final del mismo.

7.1 Planificación

A continuación se muestra la planificación del proyecto, en la que se estima una duración de 12 meses:

Nombre de tarea	Duración	Comienzo	Fin
Análisis	**75 días**	**mié 01/09/10**	**mar 14/12/10**
Sistema ficheros	24 días	mié 01/09/10	lun 04/10/10
Base de datos	18 días	lun 04/10/10	mié 27/10/10
Comunicación	18 días	mié 27/10/10	vie 19/11/10
Fuse	18 días	vie 19/11/10	mar 14/12/10
Diseño	**66 días**	**vie 10/12/10**	**vie 11/03/11**
Base de datos	15 días	vie 10/12/10	jue 30/12/10
Comunicación	15 días	jue 30/12/10	mié 19/01/11
Fuse	14 días	mié 19/01/11	lun 07/02/11
Integracion	13 días	lun 07/02/11	mié 23/02/11
Funciones basicas	13 días	mié 23/02/11	vie 11/03/11
Implementación	**77 días**	**jue 10/03/11**	**vie 24/06/11**
Base de datos	15 días	jue 10/03/11	mié 30/03/11
Comunicación	20 días	mié 30/03/11	mar 26/04/11
Fuse	15 días	mar 26/04/11	lun 16/05/11
Otras funciones	30 días	lun 16/05/11	vie 24/06/11
Pruebas	**29 días**	**lun 20/06/11**	**jue 28/07/11**
Verificacion	10 días	lun 20/06/11	vie 01/07/11
Rendimiento	20 días	vie 01/07/11	jue 28/07/11
Documentacion	29 días	lun 25/07/11	jue 01/09/11

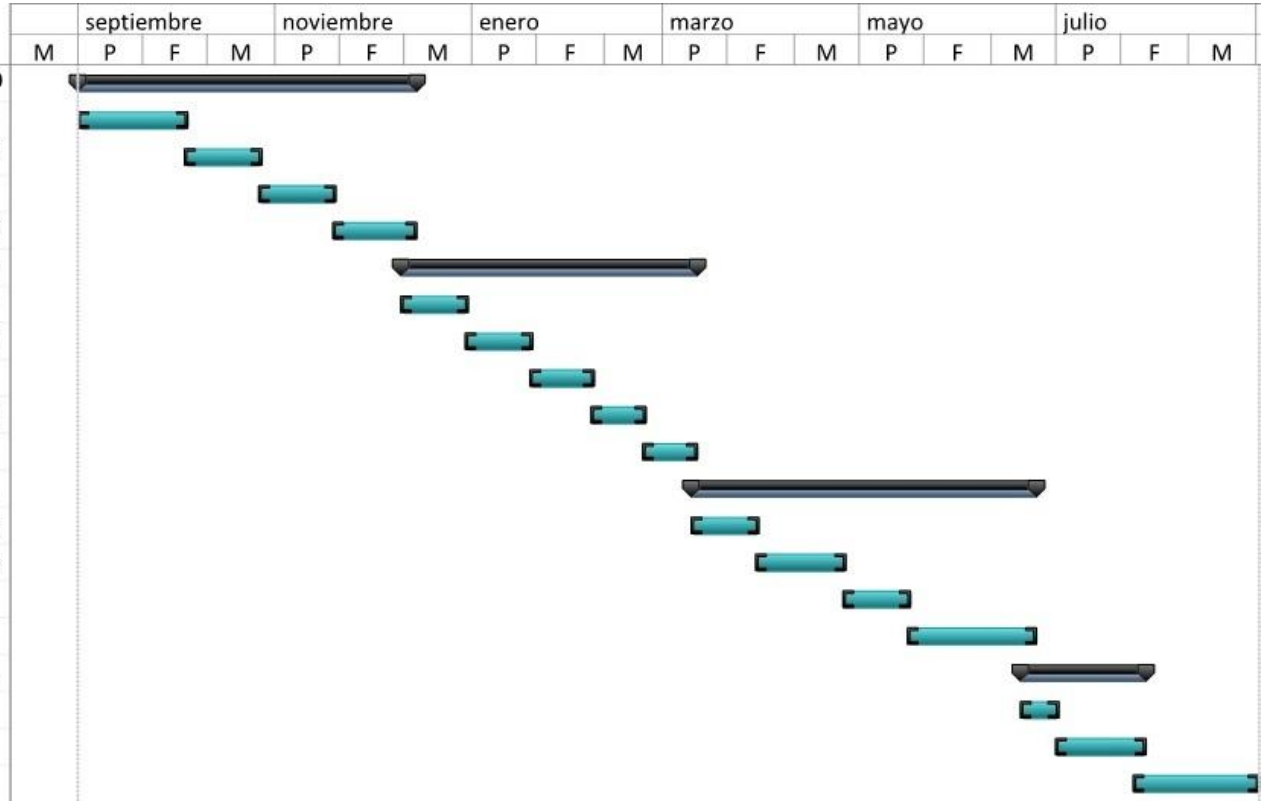


Diagrama de Gantt 1. Planificación del proyecto

7.2 Presupuesto

En esta sección se detalla el presupuesto final del proyecto.

7.2.1 Coste personal

En la siguiente tabla se detalla el cálculo del coste por hora de los diferentes roles que intervinieron en el proyecto.

	Jefe de proyecto	Analista	Diseñador
Días laborables / mes	20 días		
Horas trabajadas / día	8 horas		
Horas trabajadas / mes	160 horas		
Salario neto / mes	2.500 €	1.500 €	1.500 €
IRPF (20% del salario neto)	500 €	300 €	300 €
Seguridad Social (40% del salario neto + IRPF)	1.200 €	720 €	720 €
Salario bruto / mes	4.200 €	2.520 €	2.520 €
Meses Laborables	11 meses		
Salario bruto / año (14 pagas)	58.800 €	35.280 €	35.280 €
Horas trabajadas / año	1760 horas		
Coste bruto por persona / hora	33,41 €	20,05 €	20,05 €

Tabla 4. Calculo coste persona/hora (I)

	Programador	Pruebas	Documentación
Días laborables / mes	20 días		
Horas trabajadas / día	8 horas		
Horas trabajadas / mes	160 horas		
Salario neto / mes	1.200 €	1.200 €	1.000 €
IRPF (20% del salario neto)	240 €	240 €	200 €
Seguridad Social (40% del salario neto + IRPF)	576 €	576 €	480 €
Salario bruto / mes	2.016 €	2.016 €	1.680 €
Meses Laborables	11 meses		
Salario bruto / año (14 pagas)	28.224 €	28.224 €	23.520 €
Horas trabajadas / año	1760 horas		
Coste bruto por persona / hora	16,04 €	16,04 €	13,36 €

Tabla 5. Calculo coste persona/hora (II)

A continuación se detallan las horas trabajadas por persona en cada fase del desarrollo del proyecto:

Actividad	Jefe de proyecto	Analista	Diseñador	Programador	Encargado Pruebas	Encargado de Documentación
Análisis	80 h	270 h	53 h	0 h	0 h	0 h
Diseño	30 h	0 h	235 h	0 h	0 h	0 h
Implementación	20 h	0 h	0 h	360 h	0 h	0 h
Pruebas	0 h	0 h	0 h	45 h	110 h	0 h
Documentación	19 h	10 h	0 h	0 h	0 h	200 h

Tabla 6. Horas trabajadas

En la siguiente tabla se describe el coste derivado del personal:

Perfil	Coste / hora	Horas trabajadas	Coste
Jefe de proyecto	33,41 €	149 horas	4.977,95 €
Analista	20,05 €	280 horas	5.612,73 €
Diseñador	20,05 €	288 horas	5.773,09 €
Programador	16,04 €	405 horas	6.494,73 €
Encargado de Pruebas	16,04 €	110 horas	1.764,00 €
Encargado de Documentación	13,36 €	200 horas	2.672,73 €
		Total	**27.295,23 €**

Tabla 7. Coste personal

7.2.2 Coste Hardware y licencias

En la siguiente tabla se desglosa el coste de hardware y licencias utilizados para el desarrollo del proyecto:

Descripción	Coste
Ordenador sobremesa	**510 €**
Procesador: Intel Dual Core E5400	170 €
Placa base: Asus P5KPL-AM IN Socket 775	150 €
Disco duro: 1TB SATA2	50 €
Memoria: Kingston 4Gb DDR2 800MHz	60 €
Tarjeta gráfica: Integrada	-
Tarjeta de sonido: Integrada	-
Fuente de alimentación: 500W	80 €
Ordenador portátil	**800 €**
Asus ul50vt	800 €
Licencias y software	**139 €**
Sistema operativo: Ubuntu 10.4	0 €
IDE: Eclipse Helios	0 €
Microsoft Office 2010 Hogar y estudiantes	139 €
Total	**1.449 €**

Tabla 8. Coste hardware y licencias

7.2.3 Coste material fungible

Detalle del coste de material fungible:

Descripción	Coste
Material oficina	**30 €**
Folios	20 €
Bolígrafos	10 €
Total	**30 €**

Tabla 9. Coste material fungible

7.2.4 Coste oficina

Desglose del coste derivado de la oficina:

Descripción	Coste
Alquiler	**365 € / mes**
Oficina	300 € / mes
Acceso a internet	25 € / mes
Luz	40 € / mes
Mobiliario	**330 €**
Escritorio	120 €
Silla oficina	210 €
Total (12 meses)	**4.710 €**

Tabla 10. Coste oficina

7.2.5 Coste final del proyecto

En la siguiente tabla se resume el coste final del proyecto:

Descripción	Coste
Coste de personal	27.295,23 €
Equipos informáticos y licencias	1.449 €
Material fungible	30 €
Oficina	4.710 €
Costes directos	33.484,23 €
Costes indirectos = 15% sobre Costes directos	5.022,63 €
Riesgo = 20% sobre Coste de personal	5.459,05 €
Total costes = Costes directos + Costes indirectos + Riesgo	43.965,91 €
Beneficio = 15% sobre Total costes	6.594,89 €
TOTAL = Total costes + Beneficio	**50.560,79 €**

Tabla 11. Coste final del proyecto

8. Bibliografía

Berkeley DB Tutorial. (s.f.). Recuperado el Octubre de 2010, de http://www.mathematik.uni-ulm.de/help/BerkeleyDB/reftoc.html

BerkeleyDB documentation. (s.f.). Recuperado el Octubre de 2010, de http://download.oracle.com/docs/cd/E17076_02/html/toc.htm

BerkelyDB. (s.f.). Recuperado el Octubre de 2010, de http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html

Bloomer, J. (1992). *Power programming with RPC.* O'Reilly Media, Inc.

Cluster Resources. (s.f.). Recuperado el Octubre de 2010, de http://www.clusterresources.com/

Coda File System. (s.f.). Recuperado el Junio de 2011, de http://www.coda.cs.cmu.edu/

FileSystem - fuse. (s.f.). Recuperado el Enero de 2011, de http://sourceforge.net/apps/mediawiki/fuse/index.php?title=FileSystems

Fuse Documentation. (s.f.). Recuperado el Enero de 2011, de http://fuse.sourceforge.net/doxygen/index.html

FUSE: Filesystem in Userspace. (s.f.). Recuperado el Enero de 2011, de http://fuse.sourceforge.net/

IBM General Parallel File System. (s.f.). Recuperado el Junio de 2011, de http://www-03.ibm.com/systems/software/gpfs/index.html

LUSTRE. (s.f.). Recuperado el Junio de 2011, de http://wiki.lustre.org/index.php/Main_Page

Parallel Virtual File System. (s.f.). Recuperado el Junio de 2011, de http://www.pvfs.org/

RPC (Remote Procedure Call). (s.f.). Recuperado el Octubre de 2010, de http://www.chuidiang.com/clinux/rpc/rpc.php

S. Ghemawat, H. a.-T. (2003). The Google File System. *19th ACM Symposium on Operating Systems Principles.* Lake George, NY.

Sanjay Ghemawat, H. G.-T. (2003). "The google file system". *Proceedings of the nineteenth ACM symposium on Operating systems principles*, (págs. 29-43).

9. Apéndice I. Manual de instalación

Instalar BerkeleyDB

1. Descargar BerkeleyDB
    1. Entrar en www.oracle.com.
    2. Hacer *clic* en “Downloads”.
    3. En el apartado “Database”, hacer *clic* en “Berkeley DB”.
    4. Descargar la última versión disponible.
2. Extraer ficheros.
    1. `tar –xvzf db-x.x.xx.tar.gz`
    2. `[En consola] cd db-x.x.xx`
3. Configurar Berkeley DB
    1. `cd build_unix`
    2. `../dist/configure --prefix=/usr/local/berkeleydb --enable-compat185 --enable-cxx --enable-debug_rop --enable-debug_wop`
4. Compilar
    1. `make`
5. Instalar
    1. `sudo make install`

Instalar Fuse

1. Descargar FUSE
    1. Entrar en http://fuse.sourceforge.net/
    2. Hacer *clic* en "Download pre-release".
    3. Hacer *clic* en "Download fuse-x.x.x.tar.gz".
2. Descomprimir el fichero.
    1. `./configure`
    2. `make`
    3. `make install`

10. Apéndice II. Configuración

Configuración IOP & Cliente

Tanto la máquina que ejecuta el nodo IOP como la máquina que ejecuta el nodo Cliente deben definir una variable de entorno, que definirá el tamaño del bloque de datos, los pasos a seguir para definir dicha variable son los siguientes:

1. Abrir una consola de comandos
2. `chmod 777 /etc/environment`
3. Abrir el fichero de texto /etc/environment
4. Añadir la línea: *LEN_BLOCK="x"* donde x es el valor del tamaño de bloque.

Configuración IOP

Ejemplo del fichero de configuración:

```
nameNetIOP hola

posIOP 1

dbType 1

dbName dbBloques

dirServer 127.0.1.1

dbSpaceGB 4
```

```
freeNumBlock 2

localPath /home/a/Escritorio/iop

blockReplacement 1

writePolitic 1

valueWritePolitic 10

readPolitic 0

valueReadPolitic 0
```

- **nameNetIOP**: nombre de la red de nodos IOP de la que formara parte el IOP
- **posIOP**: posición dentro de la red de nodos IOP que ocupara el IOP.
- **dbType**: tipo de base de datos.
    - **1** – en memoria
    - **2** – en disco
- **dbName**: nombre de la base de datos en disco. Solo es necesario si dbType vale 2.
- **dirServer**: dirección ip del nodo de Metadatos
- **dbSpaceGB:** espacio máximo que puede ocupar la base de datos.
- **freeNumBlock:** numero de bloques que se desean liberar cuando la base de datos se llene.
- **localPath:** ruta del directorio donde se encuentran los ficheros.
- **blockReplacement:** política de reemplazamiento de bloques.
    - **0 –** primero se borran los bloques sucios y luego los limpios.
    - **1 –** primero se borran los bloques limpios y luego los sucios.
- **writePolitic:** política de escritura.
    - **0 – INSTANT_UPDATE** los bloques de escrituras se vuelcan directamente a disco.
    - **1 – N_IOP** los bloques de escritura se copian a X nodos IOP vecinos. El número de X es el valor de valueWritePolitic.
    - **2 – UPDATE_TIME** los bloques se vuelcan a disco cada X segundos. El número de X es el valor de valueWritePolitic.
- **valueWritePolitic:** explicado en el punto anterior.
- **readPolitic:** política de lectura.

    - **0 – ONLY_DEMAND** se cargan en la DB los bloques de lectura que pide el cliente.
    - **1 – NEXTS_BACKGROUND** se cargan en la DB los bloques de lectura que pide el cliente, y posteriormente en segundo plano se cargan los X siguientes bloques al solicitado. El número de X es el valor de valueReadPolitic.
    - **2 – NEXT_PASSIVE_WAITING** se carga en la DB el bloque solicitado por el cliente y los X siguientes. El número de X es el valor de valueReadPolitic.
- **valueReadPolitic:** explicado en el punto anterior.

Configuración Cliente

Ejemplo del fichero de configuración:

```
dirServer 127.0.1.1

reconnectionTime 1

localPath /home/a/Escritorio/iop

iopNetName hola
```

- **dirServer**: dirección ip del nodo DNS
- **reconnectionTime**: tiempo en segundos que debe transcurrir tras detectar la caída de un nodo IOP para volver a intentar conectarse.
- **localPath**: ruta del directorio donde se encuentran los ficheros.
- **iopNetName**: nombre de la red de nodos IOP a los que se va a conectar el cliente.

11. Apéndice III. Manual de usuario

Compilación Modo DEBUG

Para mostrar la información del modo debug se debe añadir -DDEBUG en todos los ficheros de la carpeta "Debug/src" de cada proyecto con extensión "*.mk".

Ejemplo debug desactivado:

```
g++ -O0 -g3 -Wall …
```

Ejemplo debug activado:

```
g++ -O0 -g3 -DDEBUG -Wall …
```

Compilación

Cada uno de los programas CLIENT, DNS y IOP tienen la misma estructura de directorios. En la carpeta principal del programa se encuentra una carpeta "src", cuyo contenido se encuentra detallado en el Apéndice IV, y la carpeta "Debug". Para compilar el programa debemos acceder a la carpeta "Debug" y ejecutar:

- `make clean`
- `make`

Ejecución

Para ejecutar el **nodo Cliente** entramos en la carpeta "CLIENT/Debug":

```
./CLIENT /ruta/config.cfg /ruta/FUSE -o direct_io
```

Ejecuta un nodo Cliente con la configuración del fichero “config.cfg” y monta el cliente fuse en el directorio “/FUSE”. Si queremos que el nodo Cliente muestre por pantalla información sobre la ejecución debemos ejecutarlo como:

```
./CLIENT /ruta/config.cfg /ruta/FUSE -d -o direct_io
```

Para ejecutar el **nodo de Metadatos** entramos en la carpeta “DNS/Debug”:

```
./DNS
```

Para ejecutar el **nodo IOP** entramos en la carpeta “DNS/Debug”:

```
./IOP /ruta/config.cfg
```

Figura 24. Orden ejecución

Comandos nodo IOP

- “clear”: limpia el contenido de la consola.
- “info”: muestra información del nodo IOP. Ver imagen 6.
  - Cache hit rate: porcentaje de acierto de la cache.
  - DB free blocks: número de bloques libres en la base de datos
  - DB used blocks: número de bloques en la base de datos.
  - DB dirty blocks: número de bloques sucios en la base de datos.
  - DB clean blocks: número de bloques limpios en la base de datos.
  - DB backup blocks: número de bloques backup en la base de datos.
  - Number of files opens: número de ficheros abiertos en el sistema.

- “infoDB”: muéstralos bloques de la base de datos. Ver imagen 6.
    - Clean Blocks: lista los bloques limpios insertados en la base de datos.
    - Dirty Blocks: lista los bloques sucios insertados en la base de datos.
    - Backup Blocks: lista los bloques de backup insertados en la base de datos.
- “enable”: habilita la impresión por pantalla.
- “disable”: deshabilita la impresión por pantalla.

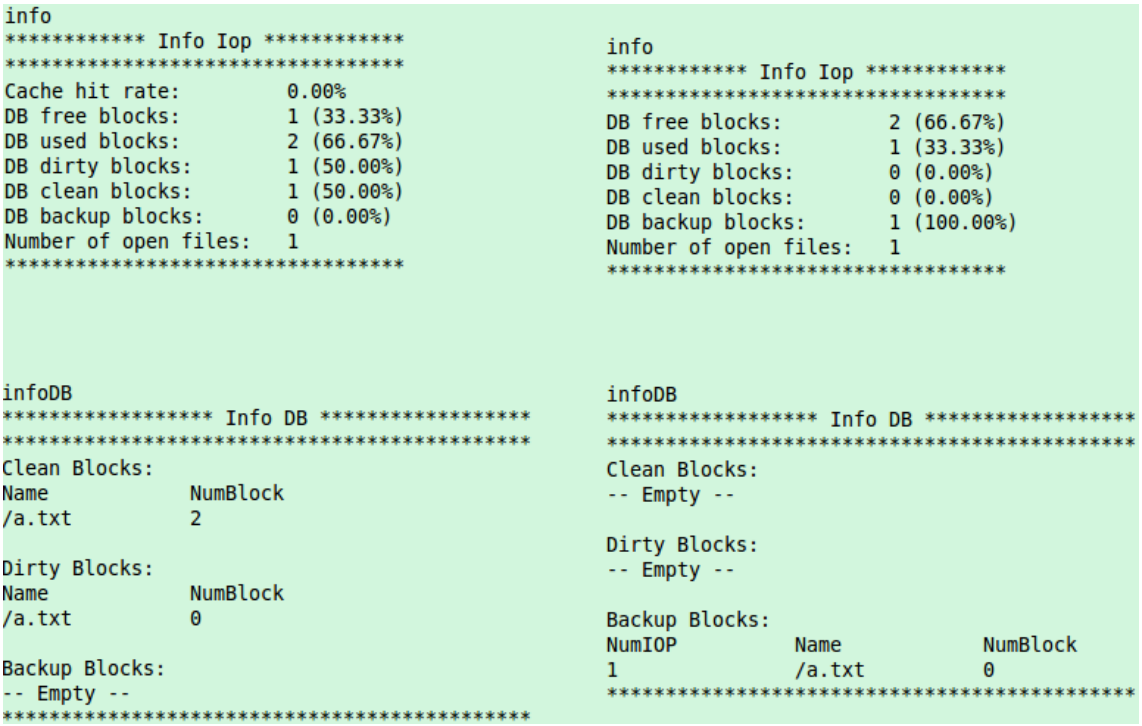


Figura 25. Comando info e infoDB

En la imagen se muestra el comando info e infoDB ejecutado en 2 nodos IOP pertenecientes a la misma red. Sobre el nodo de la derecha se ha ejecutado una lectura y una escritura y ha realizado una copia de la escritura en el nodo de la derecha.

12. Apéndice IV. Ficheros código fuente

Ficheros comunes a los 3 sistemas:

***_clnt.c:** Autogenerado mediante la herramienta rpcgen, contiene las funciones de las llamadas rpc.

***_xdr.c:** Autogenerado mediante la herramienta rpcgen, contiene la traducción de los tipos de datos al estándar XDR.

print.cpp: contiene las funciones para imprimir en modo árbol, similar a un modo *Debug* jerárquico.

print.h: cabecera de la clase print.cpp

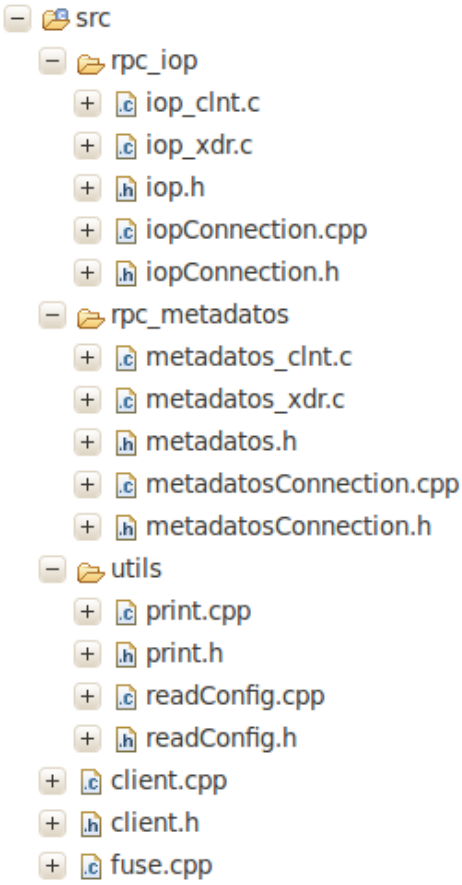


Figura 26. Estructura ficheros nodo Cliente

dns.h: autogenerado mediante la herramienta rpcgen. Contiene los prototipos de las funciones y las estructuras de las llamadas rpc.

dnsConnection.cpp: realiza las llamadas rpc y gestiona la reconexión automática con el servidor dns.

iop.h: autogenerado mediante la herramienta rpcgen. Contiene los prototipos de las funciones y las estructuras de las llamadas rpc.

iopConnection.cpp: realiza las llamadas rpc y gestiona la reconexión automática con el servidor iop.

readConfig.cpp: lee el fichero de configuración del cliente, y almacena los valores en variables.

client.cpp: contiene el código de las llamadas FUSE open, release, write y read.

fuse.cpp: contiene las llamadas FUSE del sistema operativo. Las llamadas modificadas (open, release, write y read) no están definidas en esta clase. Main del programa.

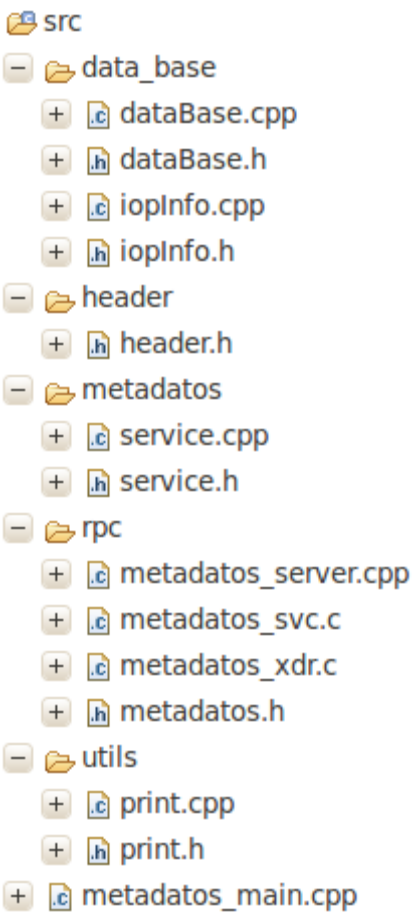


Figura 27. Estructura ficheros nodo de Metadatos

dataBase.cpp: base de datos, contiene las funciones abrir db, insertar dato y borrar dato.

iopInfo.cpp: convierte la información para la base de datos.

service.cpp: contiene la implementación de las llamadas rpc.

header.h: contiene las estructuras de datos.

dns_server.cpp: Contiene el código de las funciones rpc que ofrece el servidor dns.

dns.h: autogenerado mediante la herramienta rpcgen. Contiene los prototipos de las funciones y las estructuras de las llamadas rpc.

dns_main.cpp: Main del programa.

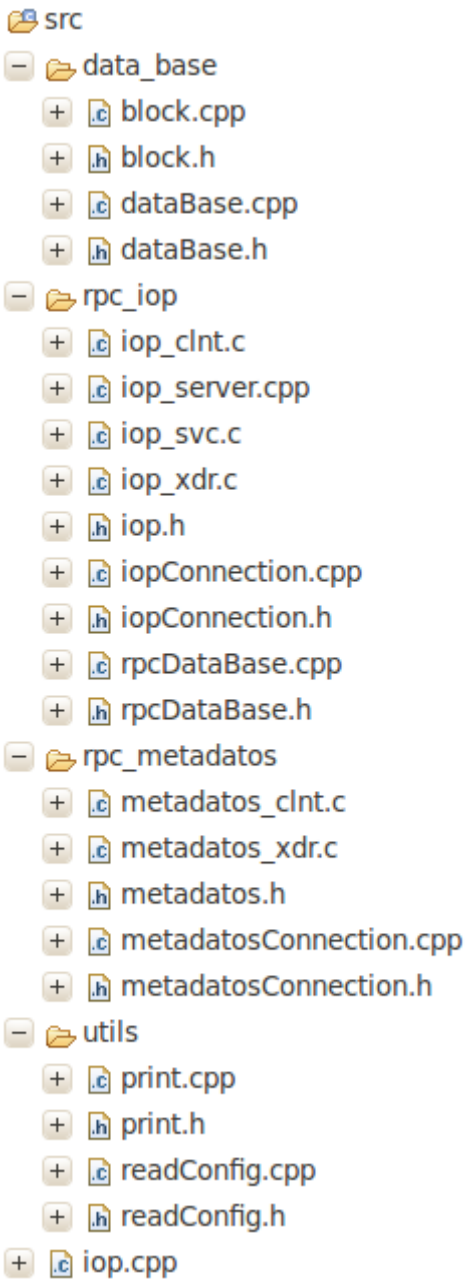


Figura 28. Estructura ficheros nodo IOP

block.cpp: convierte la información para la base de datos.

dataBase.cpp: base de datos, contiene las funciones abrir db, insertar dato y borrar dato.

dns.h: autogenerado mediante la herramienta rpcgen. Contiene los prototipos de las funciones y las estructuras de las llamadas rpc.

dnsConnection.cpp: realiza las llamadas rpc y gestiona la reconexión automática con el nodo de Metadatos.

iop.h: autogenerado mediante la herramienta rpcgen. Contiene los prototipos de las funciones y las estructuras de las llamadas rpc.

iopConnection.cpp: realiza las llamadas rpc y gestiona la reconexión automática con el servidor iop.

rpcDataBase.cpp: contiene la implantación de las llamadas rpc que ofrece el iop.

readConfig.cpp: lee el fichero de configuración del cliente, y almacena los valores en variables.

iop.cpp: Main del programa.

13. Apéndice V. Manual print

La clase print sirve para imprimir mensajes en cascada, facilitando seguir la traza de llamadas anidadas.

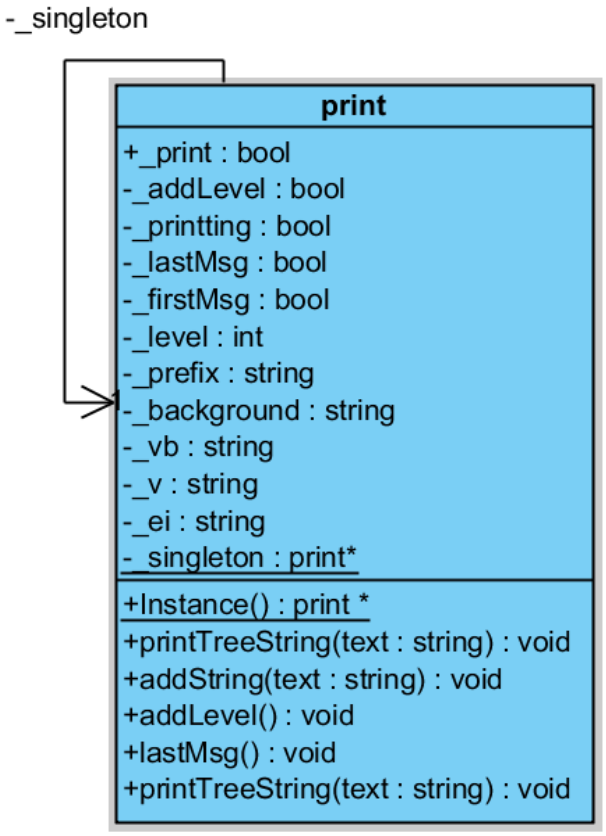


Figura 29. Clase print

Funciones:

- printTreeString: imprime un texto en forma de árbol jerárquico en pantalla.
- addLevel: añade un nivel de profundidad, el siguiente mensaje que se imprima en pantalla se hará en el nuevo nivel.
- lastMsg: indica que es el último mensaje del nivel, el siguiente mensaje que se imprima en pantalla se hará en el nivel anterior.
- addString: añade un texto para imprimir que se imprimirá cuando se termine el árbol de impresión actual o en caso de que no se esté imprimiendo ningún árbol se imprimirá automáticamente en pantalla.
- enablePrint: habilita la salida por pantalla.
- disablePring: deshabilita la salida por pantalla.

Ejemplo:

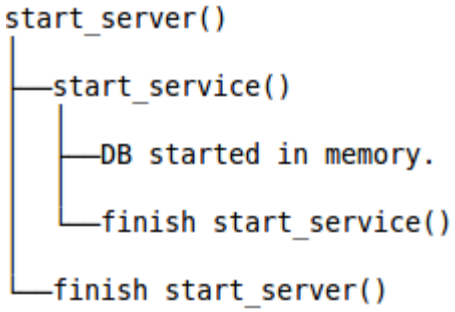


Figura 30. Ejemplo print

Para lograr la salida por pantalla de la imagen 8, se ejecutarían las siguientes llamadas:

- printTreeString(“start_server()”)
- addLevel()
- printTreeString(“start_service()”)
- printTreeString(“DB started in memory”)
- lastMsg()
- printTreeString(“finish start_service()”)
- lastMsg()
- printTreeString(“finish start_server()”)

14. Apéndice VI. Callgrind y Kcachegrind

Callgrind es una herramienta de análisis que registra el historial de llamadas entre las funciones de un programa en un fichero de texto plano. Por defecto, los datos recogidos consisten en el número de instrucciones ejecutadas, su relación con las líneas de código y la relación de llamadas entre las funciones.

Kcachegrind es una herramienta que permite dado un informe generado mediante la herramienta Callgrind generar un grafico.

A continuación se muestran los gráficos de ejecución del inicio de cada uno de los 3 tipos de nodo. Los gráficos muestran el desglose en porcentajes del tiempo empleado para ejecutar cada subrutina.

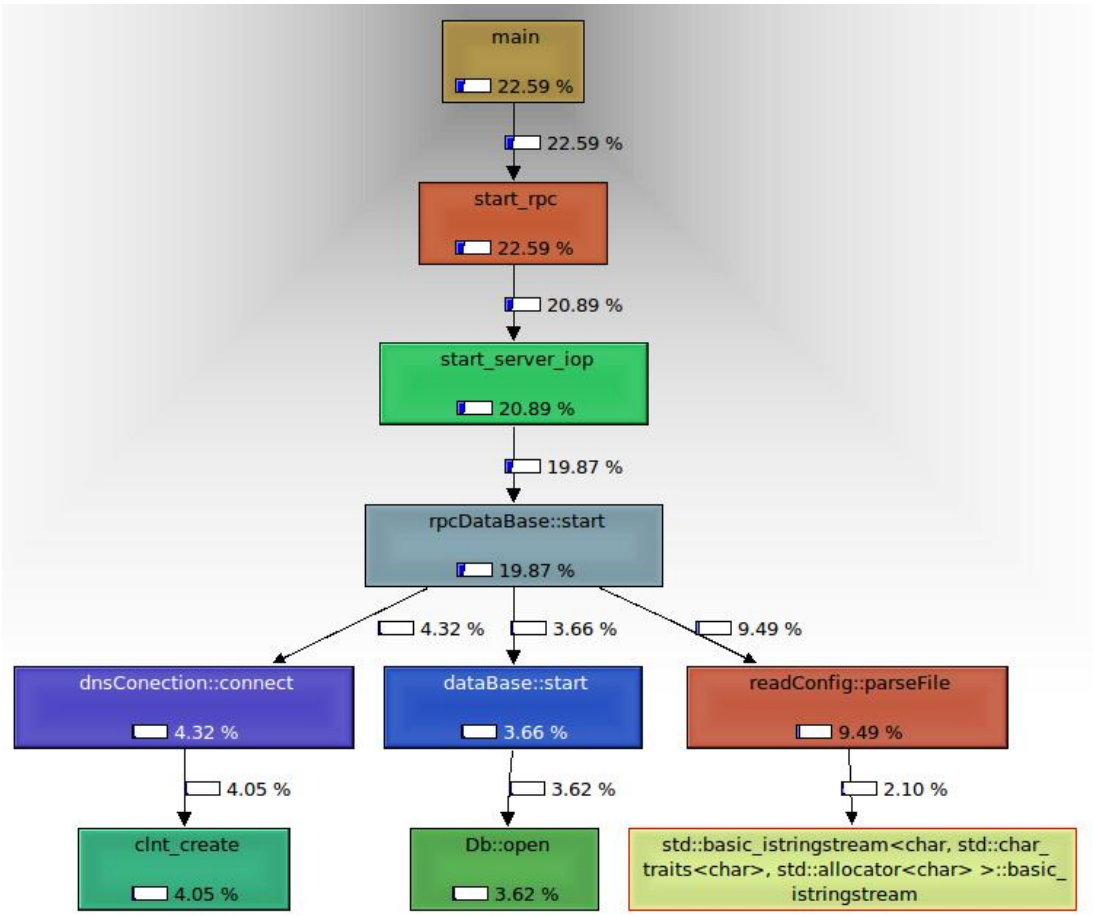


Figura 31. Kachegrind inicio nodo IOP

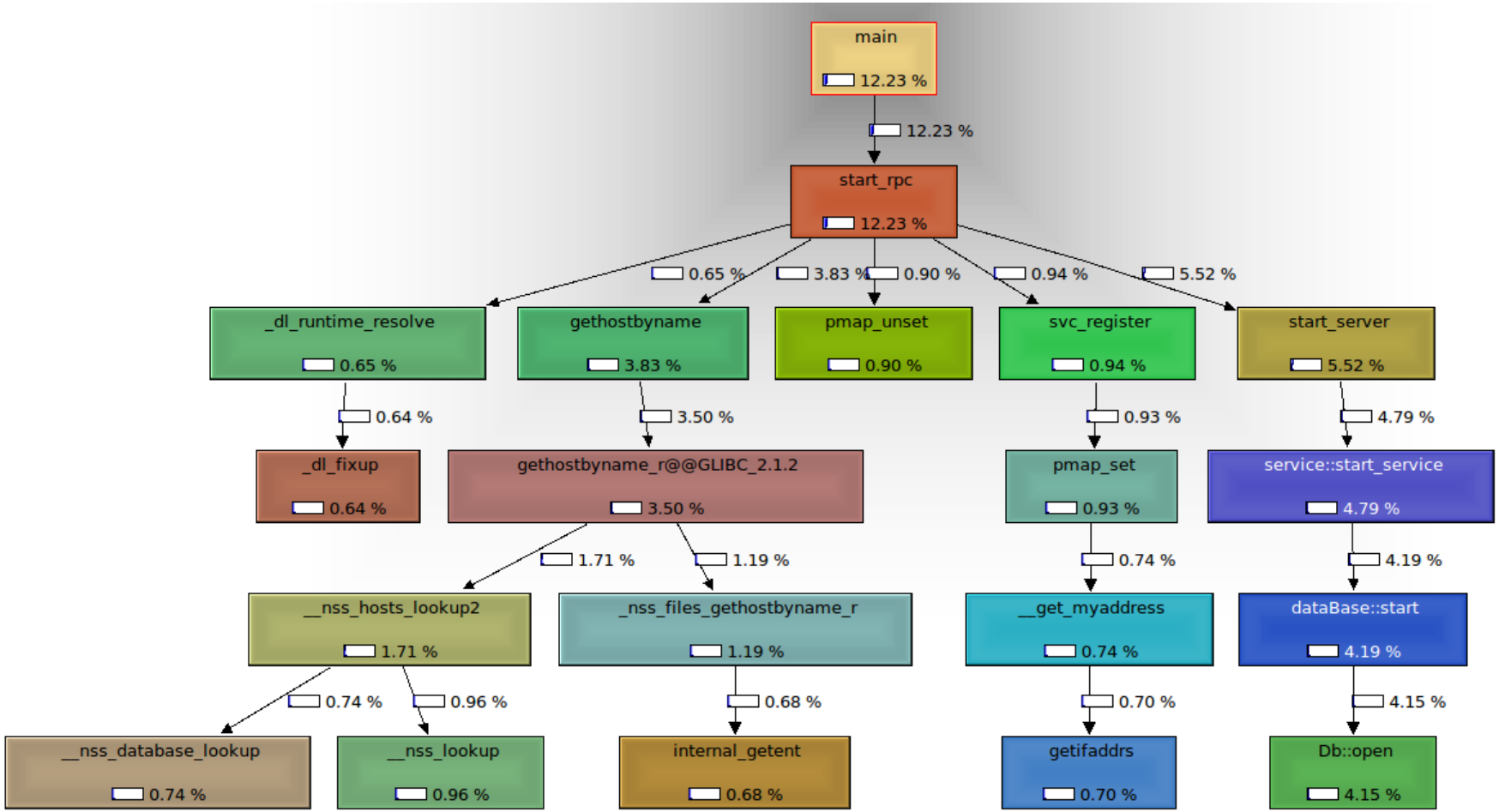


Figura 32. Kachegrind inicio nodo de Metadatos

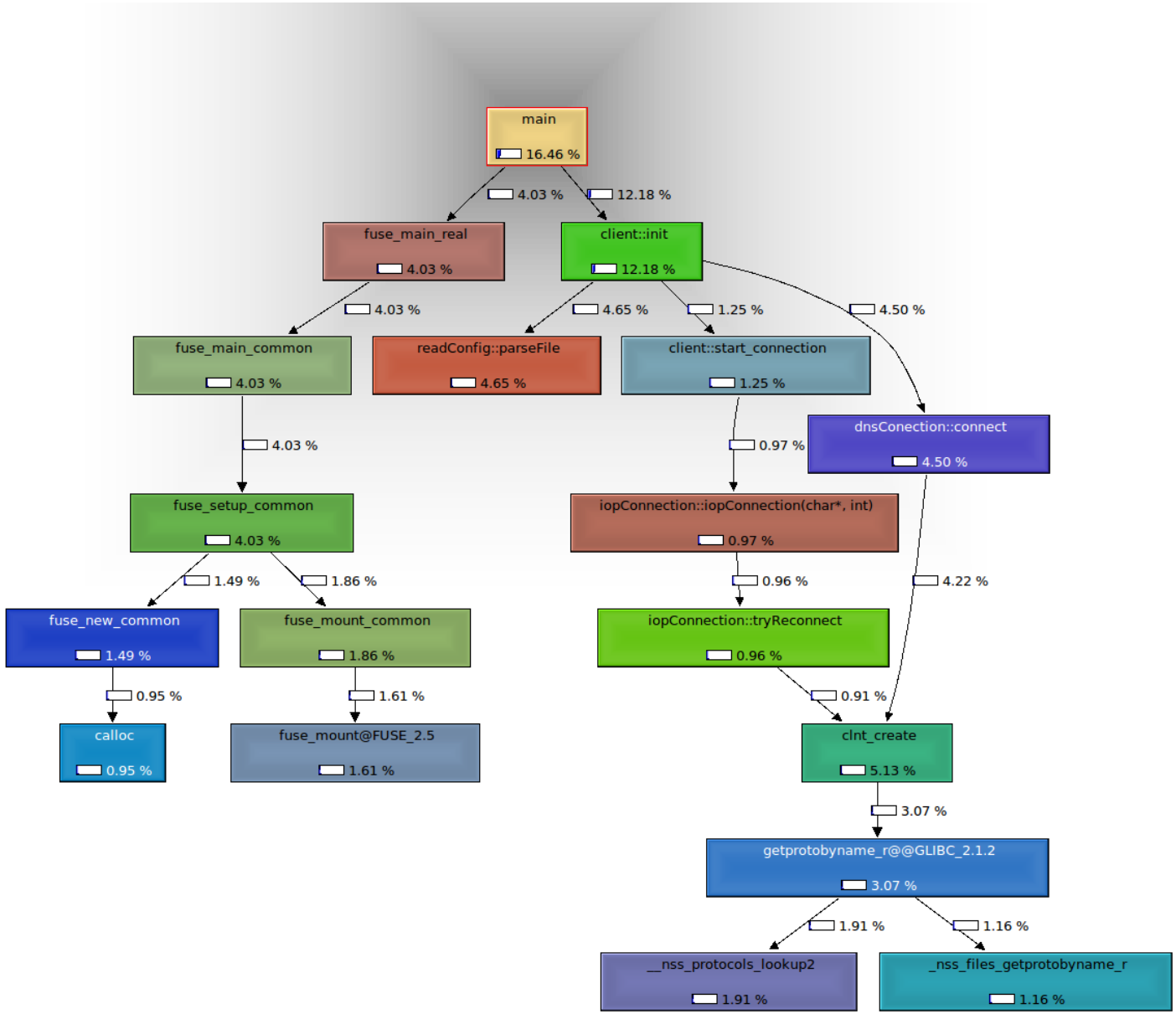


Figura 33. Kachegrind inicio nodo Cliente